Nichtlineare adaptive Regelung mit neuronalen Netzen im unbemannten Flugversuch

Von der Fakultät für Maschinenbau
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung der Würde

eines Doktor-Ingenieurs (Dr.-Ing.)

genehmigte Dissertation

von:	Karl Kufieta
aus:	Loslau

eingereicht am:	09.01.2015
mündliche Prüfung am:	23.04.2015

Gutachter:	Prof. Dr.-Ing. Peter Vörsmann
	Prof. Dr.-Ing. habil. Gert F. Trommer

2015

Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Luft- und Raumfahrtsysteme der Technischen Universität Braunschweig in den Jahren 2010 bis 2014.

An erster Stelle möchte ich mich bei Herrn Prof. Dr.-Ing. Peter Vörsmann für das mir entgegengebrachte Vertrauen, Interesse und die mir eingeräumten Freiheiten im Rahmen meiner Forschungstätigkeiten bedanken. Ebenso danke ich Prof. Dr.-Ing. Gert Trommer für das Interesse an dieser Arbeit, die hilfreichen Ratschläge und für die Übernahme des Korreferates.

Viel Dank gilt auch allen Mitarbeitern des Instituts für Luft- und Raumfahrtsysteme. Das gute Gelingen dieser Arbeit wäre ohne das kollegiale Umfeld und im Besonderen den kritischen Diskussionen mit den Arbeitsgruppen *Meteorologie* und *Unbemannte Flugsysteme* nicht möglich gewesen.

Ganz besonderer Dank gilt meiner Familie, allen voran meiner Freundin Melanie, meinem Vater Paul sowie meiner Mutter Sylvia, die mich während dem Erstellen dieser Arbeit und auch während meiner gesamten universitären Laufbahn nach Kräften unterstützt haben.

Ostfildern im Dezember 2014 Karl Kufieta

Kurzfassung

Die vorliegende Arbeit stellt einen Beitrag zur praktischen Anwendung adaptiver nichtlinearer Regler in unbemannten Flugzeugen dar. Dabei wird die erstmalige Flugerprobung eines nichtlinearen adaptiven Reglers in Form der dynamischen Inversion mit Sliding-Mode-Control trainierten neuronalen Netzen auf einem sequentiellen Smartphone-Prozessor betrachtet.

Neben dem nichtlinearen Regler kommt ein Kalman Filter und ein Bahnregler zum Einsatz, um ein automatisches Abfliegen einer Flugbahn zu ermöglichen. Das wiederholte Abfliegen der Flugbahn ermöglicht den Vergleich zwischen einer konservativen, nichtlinearen und nichtlinearen adaptiven Regelungstechnik. Um einen Vergleich zwischen dem nichtlinearen Regler mit und ohne adaptive Elemente zu ermöglichen, wird zudem mit dem Reference-Control-Hedging eine Variation des bekannten Pseudo-Control-Hedging vorgestellt.

Diese, auf einem Echtzeitbetriebssystem laufenden und mit Autocode-Verfahren programmierten Algorithmen, werden neben deren Verhalten im Flugversuch auch im Hinblick auf Ressourcenverbrauch und Verzögerungen untersucht. Die Verwendung von Autocode-Verfahren erlaubt eine schnelle Implementierung der Algorithmen direkt aus einem digitalen, auf Piktogrammen basierten Schaltplan. Die Vermutung, dass diese Implementierungstechnik die Entwicklungsgeschwindigkeiten im Rahmen von Regler- und Navigations-Prototypenversuchen beschleunigen können, wird durch die Betrachtung der Geschichte der Implementierung bekräftigt.

Die Untersuchungen erlauben eine Abschätzung des Potenzials adaptiver Regler-Navigations-Strukturen und der nötigen bzw. verfügbaren Ressourcen moderner Smartphone-Prozessoren. Die Ergebnisse der Flugversuche bestätigen die in der Simulation vorhergesagten Fähigkeiten der adaptiven Strukturen. Gleichzeitig wird mit der Vielzahl an verfügbaren Algorithmen und deren Rekombinationsmöglichkeiten und der direkten Implementierbarkeit aus einem digitalen Schaltplan deutlich: Die Vereinheitlichung der Flugexperimente und der Paradigmenwechsel des Regelungstechnikers vom Programmierer hin zum Designer, kann die Vergleichbarkeit verbessern und bietet die Möglichkeit, das Potenzial der Rekombinationsmöglichkeiten auf einer neuen Implementierungsebene auszuschöpfen.

Abstract

The present work is a contribution to the practical application of an adaptive nonlinear controller in unmanned aircraft. In this case the first time flight testing of a nonlinear adaptive controller is considered in the form of dynamic inversion with Sliding-Mode-Control trained neural networks on a sequential smartphone processor.

In addition to the nonlinear controller, a Kalman filter and a path controller is used to enable an automatic trajectory following. The repeated flying of the flight path allows the comparison between a conservative, a nonlinear and a nonlinear adaptive control technique. In order to allow a comparison between the nonlinear controller with and without adaptive elements, Reference-Control-Hedging as a variation of the well known Pseudo-Control-Hedging is presented.

These algorithms, running on a real-time operating system and programmed with autocode, are studied alongside their behavior in flight tests, in terms of resource consumption and delays. The used autocode-method allows a fast implementation of the algorithms directly from a digital schematic diagram, which is based on pictograms. The assumption, that this implementation technique can accelerate the speed of development in the context of controller and navigation prototype testing, is strengthen by looking at the history of implementation.

The studies allow an estimate of the potential of adaptive controller-navigation structures and the necessary and available resources of modern smartphone processors. The results of the flight tests confirm the capabilities of adaptive structures, which have been predicted in simulations. The number of available algorithms and their recombination possibilities and the possibility of direct implementability from a digital schematic diagram shows: The standardization of flight experiments can improve the comparability of the results. Combined with the paradigm shift of the control engineer, from a programmer to a designer, this new implementation level can make the recombination possibilities explorable.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

[ARM]	*advanced reduced instruction set computers*
[AVL]	*Athena vortex lattice*
[BINF]	Byrnes-Isidory-Normalform
[EKF]	*extended Kalman filter*
[FPGA]	*field-programmable gate array*
[GPIO]	*general purpose input output*
[HIL]	*hardware in the loop*
[HR]	*high resolution*
[IMU]	*inertial measurement unit* - inertiale Messplattform
[KNN]	künstliches neuronales Netzwerk
[MIMO]	*multiple input multiple output* - Mehrgrößensystem
[MRAC]	*model reference adaptive control*
[NDI]	*nonlinear dynamic inversion*
[PCH]	*pseudo control hedging*
[RCH]	*reference control hedging*
[SISO]	*single input single output* - Eingrößensystem
[SMC-BP]	*sliding mode control backpropagation*
[SPI]	*serial peripheral interface*
[SPIO]	SPI bzw. SPI Treiber in Linux
[TLC]	*target language compiler*
[UAV]	*unmanned aerial vehicle* - unbemanntes Luftfahrzeug
[VHDL]	*very high speed integrated circuit hardware description language*
[WGS84]	*world geodetic system 1984*

Formelzeichen

Lateinische Buchstaben

Symbol	Einheit	
$\mathbf{A}(\vec{x})$	–	Entkopplungsmatrix
$\mathbf{A}_E$	–	Systemmatrix der Fehlerdynamik
$a(\vec{\xi}, \vec{\eta})$	div.	Element der Entkopplungsmatrix
a_i	–	Koeffizienten des Referenzmodells
$\vec{a}$	m/s^2	Beschleunigungsvektor
a	–	Substitution in der SMC-BP
$\mathbf{b}_E$	–	Eingangsvektor des Inversionsfehlers
$b(\vec{\xi}, \vec{\eta})$	div.	Lie-Ableitung entlang der Ausgangsfunktion
$b^{(l)}$	-	Bias einer Neuronenschicht (l)
b	m; -	Spannweite; bzw. Substitution in der SMC-BP
C	-	aerodynamischer Beiwert
c	-	Bezugsflügeltiefe
$\vec{c}, c_i$	-	(vektorielle) Reglerparameter der Fehlerdynamik
d	m	Abstand
E	-	Netzfehler
$\vec{e}, e$	div.	(vektorieller) Regelfehler
$\mathbf{F}$	div.	nichtlineares System
$\vec{F}$	N	Kraftvektor
$\vec{f}$	div.	Vektorfeld einer nichtlinearen Abbildung
$\mathbf{G}$	div.	Abbildungsmatrix der nichtlinearen Stelldynamik
$\vec{g}$	div.	Abbildungsvektorfeld der nichtlinearen Stelldynamik
g	9,81· m/s^2	Erdbeschleunigung
$\vec{h}, h$	div.	(vektorielle) nichtlineare Ausgangsfunktion
I	kg·m^2	Trägheitsmoment
J	-	Jacobimatrix der Neuronen-Verbindungsgewichte
$\mathbf{K}$	-	Matrix mit Verstärkungsfaktoren
$\vec{K}$	-	Verstärkungsfaktorenvektor
k_{r0}, k_{r1}	-	Parameter des robustifizierenden Terms
L	N/m	x-Komponente eines Moments
L_f, L_g	-	Lie-Ableitung bezüglich $\vec{f}$, bzw. $\vec{g}$
$\mathbf{M}$	-	Rotationsmatrix

Symbol	Einheit	
$\vec{M}, M$	N/m	Momentenvektor; bzw. y-Komponente eines Moments
N	N/m	z-Komponente eines Moments
n	-	Ordnung eines Systems
$o_j^{(l)}$	-	Ausgang eines Neurons j der Schicht (l)
$\vec{P}$	[°,°]	Punkt auf der Erdoberfläche
$\vec{p}$	[°,°]	Punkt auf der Erdoberfläche
p	°/s	Rollrate
$\vec{Q}$	N/m	Momentenvektor
$\vec{q}$	−; div.	Quaternionenvektor; bzw. interne Dynamik
q_i	−	Quaternionenvektorkomponente
$\bar{q}$	N/m^2	Staudruck
q	°/s	Nickrate
$\vec{R}$	N	aerodynamischer Kraftvektor
$\vec{r}$	-	vektorieller relativer Grad
r	°/s; -	Gierrate; bzw. relativer Grad
$\vec{S}(\vec{x})$	-	Vektor von Schaltfunktionen
$\vec{S}$	div.	Vektor von Skalenfaktoren
S	m^2; −	Bezugsflügelfläche; bzw. Schaltfunktion
$\vec{s}$	div.	Position im dreidimensionalen Raum
$s^{(l)}(x^{(l)})$	-	Aktivierungsfunktion einer Schicht (l)
$\mathbf{T}$	kg·m^2	Trägheitstensor
T	°C; -	Temperatur; bzw. Konstante
t	s	Zeit
$\vec{u}$	div.	Eingangsgrößenvektor
u	div.; m/s	Eingangsgröße; bzw. x-Komp. einer Geschwindigkeit
$\vec{V}$	m/s	Geschwindigkeitsvektor
V_{TAS}	m/s	echte Fluggeschwindigkeit
v	m/s	y-Komponente einer Geschwindigkeit
$\mathbf{w}$	-	Matrix der Neuronen-Verbindungsgewichte
$w_{i,j}^{(l)}$	-	Verbindungsgewicht einer Schicht (l) zw. zwei Neuronen
w	m/s	z-Komponente einer Geschwindigkeit
X	N	x-Komponente einer Kraft
$x_i^{(l)}$	-	Eingang einer Aktivierungsfunktion einer Schicht (l)
$\underline{\vec{x}}$	-	Eingangsvektor eines neuronalen Netzes
$\vec{x}$	div.	Zustandsvektor
x	div.	x-Komponente einer Position; bzw. Zustand

Symbol	Einheit	
Y	N	y-Komponente einer Kraft
$\vec{y}$	div.	Ausgangsvektor
y	div.	y-Komp. einer Position; bzw. Ausgang eines Systems
y_{ref}	div.	Ausgang eines Referenzsystems
$y_{m,ref}$	div.	Messpunkt m eines Referenzsystems
Z	N	z-Komponente einer Kraft
$\vec{z}$	div.	Zustandsvektor der Byrnes-Isidory-Normalform
z	div.	z-Komponente einer Position

Griechische Buchstaben

Symbol	Einheit	
$\vec{\alpha}(\vec{x})$	div.	Vektor der linearisierenden Zustandsrückführung
α	$^\circ$	Anstellwinkel
$\vec{\beta}(\vec{x})$	div.	Vektor der linearisierenden Zustandsrückführung
β	$^\circ$	Schiebewinkel
$\vec{\chi}$	div.	Vektor des Regelfehlers und Regelfehlerableitungen
$\vec{\Delta}, \Delta$	$div.$	(vektorieller) Inversionsfehler
$\vec{\eta}$	div.	Zustandsvektor der internen Dynamik
ϵ	-	Netzfehler
ϵ_A	-	Restfehler
η	$^\circ$	Höhenruderwinkel
λ	-	Steigung einer Schaltlinie
μ_K	$^\circ$	Bahnhängewinkel
μ	-	Lernrate
$\vec{\nu}, \nu$	div.	(vektorielle) Pseudosteuergröße oder Ersatzregelgröße
ν_{ad}	div.	KNN Anteil der Pseudosteuergröße
ν_h	div.	Hedge-Signal des PCH
ν_r	div.	robustifizierender Anteil der Pseudosteuergröße
$\vec{\Omega}$	$^\circ$	Eulerwinkel nach LN9300
$\vec{\omega}$	$^\circ/s$	Drehratenvektor
Φ	$^\circ$; div.	Rollwinkel bzw. nichtlineare Funktion der BINF
Ψ	$^\circ$	Gierwinkel
Θ	$^\circ$	Nickwinkel
$\vec{\xi}$	div.	Zustandsvektor der E/A-Dynamik
ξ	$^\circ$	Querruderwinkel
ζ	-	gefilterte Fehlergröße

Indizes

Symbol	
A	aerodynamisch
ac	Vorsteuerung
F	Triebwerk (Kraft, Moment)
f	flugzeugfestes Koordinatensystem
g	geodätisches Koordinatensystem
I	Integralanteil
K	körperfestes Koordinatensystem
k	kommandierte Größe
$meas$	Messgröße
P	Proportionalanteil
p	bzgl. Rollrate
q	bzgl. Nickrate
r	bzgl. Gierrate
(r)	r-te Ableitung mit relativem Grad r
ref	Referenzgröße
$soll$	Sollgröße oder kommandierte Größe
TW	Triebwerk
W	Wind
Φ	bzgl. Rollwinkel
Θ	bzgl. Nickwinkel
Ψ	bzgl. Gierwinkel
Ω	bzgl. Lage
$\vec{\omega}$	bzgl. Drehratenvektor

1 Einleitung

Der Traum des Menschen von der Erschaffung künstlicher Intelligenz inspiriert seit Anbeginn der ersten Automaten die menschliche Vorstellungskraft. Der beeindruckende Fortschritt der Computertechnologie in den letzten Dekaden erlaubt erstmals in unserer Geschichte die Verwendung intelligenter Systeme in einem breiten Spektrum von Anwendungen. Intelligenz ist allerdings ein weit dehnbarer Begriff und wird für einfachste Automaten, die auf Zustandsänderungen reagieren, bis hin zum futuristischen Androiden, der bisweilen sein Dasein in Science-Fiction Romanen fristet, genutzt. Führt man diesen Begriff auf den lateinischen Ursprung zurück, so lässt sich dieser sinngemäß mit „verstehen“ und wörtlich mit „wählen zwischen“ (lat. intellegere) übersetzen. Überträgt man diese Zusammenhänge auf den Bereich der Regelungstechnik, so sollte die Intelligenz mindestens in der Lage sein, einen kausalen Zusammenhang zwischen Actio und Reactio zu erkennen und selbstständig zur Lösung eines (Regelungs-) Problems beitragen. Diese Art der Problemlösung lässt sich gut in den Begriff der Adaption einordnen.

Starre Reglerstrukturen erfassen messbare Zustände und führen diese nach klaren Regeln in das System zurück. Verändern sich die Rahmenbedingungen des Systems in unvorhergesehenem Maße, so kommen die starren Strukturen schnell an ihre Grenzen. Adaptive Strukturen hingegen besitzen die Fähigkeit zur Anpassung an veränderte Bedingungen. Dabei wählen diese selbstständig aus den zur Verfügung stehenden Messdaten jene Größen aus, die zur sinnvollen Rückführung und damit zur Verbesserung der Reglereigenschaften führen. Die Erweiterung eines Reglers um adaptive Eigenschaften bildet somit einen wichtigen Beitrag zur Translation vom Automaten hin zum autonomen System.

Die Miniaturisierung der Computersysteme erlaubt zunehmends die Entwicklung kleiner unbemannter Systeme für vielfältige Aufgaben. Die Bedeutung unbemannter Systeme in der Luft- und Raumfahrttechnik wird spätestens seit der Erforschung fremder Planeten deutlich. Unbemannte Flugsysteme finden längst Anwendung in Missionen, deren Gefahrenpotenzial und Belastung für Menschen als hoch eingestuft werden: Überwachung von havarierten Atomkraftwerken, Messungen an Vulkanen oder die Prüfung von Hochstrommasten können als Beispiele heute üblicher Einsatzgebiete genannt werden. Inzwischen wird der weite Einsatz unbemannter Flugsysteme im Logistikwesen diskutiert. Diese Entwicklungen bilden mit nachfolgenden

Zusammenhängen die Notwendigkeit adaptiver Strukturen. Während in bemannten Flugzeugen der Pilot im Schadensfall korrigierend einwirken kann, wird in der unbemannten Situation eine gewisse Handlungsautonomie zur Kompensation vorausgesetzt. Die eigenständige Wiederherstellung von Flugeigenschaften in Ausfallsituationen bildet somit einen Beitrag zur Erhöhung der Sicherheit unbemannter Systeme.

Obwohl Simulationen von Flugversuchen heutzutage einen hohen Realitätsgrad aufweisen, wird die praktische Umsetzbarkeit meist erst mit einem realen Prototypenversuch anerkannt. Bereits in den 1970er-Jahren wurden erste Flugversuche mit digitalen Systemen in der Luft- und Raumfahrttechnik, z.B. dem Kalman Filter, durchgeführt. Zu den Anfängen der Forschung an Navigations- und Regelungscomputern wurden noch einfache 8085 Prozessoren oder ein 11/34 Prozessor [Lin90, Deh83] verwendet. Bedingt durch die Dimensionen dieser Computer wurde die Peripherie wie RAM, Stromversorgung, digitale und analoge Schnittstellen in großen Schränken verbaut. Diese konnten daher ein ganzes Flugzeug füllen. Die Komplexität der Anwendung war anfangs noch stark beschränkt, z.B. auf einfache Filter oder PID-Regler. Obwohl heutige Prozessoren die vielfache Rechenleistung damaliger Systeme innehaben, sind diese Komplexitätsgrenzen allerdings nicht nur auf die Leistungsfähigkeit der Prozessoren zurückzuführen. Die Werkzeuge zur Umsetzung von Algorithmen haben sich parallel zur Computertechnik weit entwickelt. So beschreibt [Deh83] im Jahr 1983, dass die Umsetzung eines Filters in der Sprache Assembler einen hohen Zeitanteil an der Entwicklung beansprucht. Während die einfachen Filter damals noch auf Papierschaltplänen entworfen wurden, bilden diese heutzutage die Grundbibliothek moderner Simulationsumgebungen und können per Drag-and-Drop aus der Bibliothek in den digitalen Schaltplan überführt werden.

Die geschichtliche Darstellung verdeutlicht, dass die Umsetzbarkeit eines adaptiven Reglers im realen Versuch von vielen Elementen abhängt. Die Anforderung der Umsetzung in einem unbemannten Flugzeug geht allerdings noch einen Schritt weiter. Neben dem adaptiven Regler selbst wird mindestens eine Navigation vorausgesetzt. Die Durchführung automatischer Flugmissionen setzt zudem eine Bahnführung voraus.

Die vorliegende Arbeit betrachtet neben der erstmaligen Flugerprobung eines nichtlinearen adaptiven Reglers mit künstlichen neuronalen Netzen und einem neuartigen Lernverfahren nach [Krü12] auch die Rahmenbedingungen, die die folgenden Untersuchungen erst ermöglichen. Wenngleich adaptive Regler mit dem Themengebiet der

Robotik assoziiert sind, können diese vielmehr in den Bereich der Kybernetik[1] eingeordnet werden. Die Messung, Verarbeitung und Rückführung von Zuständen bildet einen stark interdisziplinären Charakter dieser Fachrichtung. Das Anwendungsgebiet der adaptiven Regler reicht somit weit über mechanische Systeme hinaus und findet auch in der Biologie, Chemie, Psychologie, Wirtschaft und allen anderen Gebieten, in denen Zustände messbar und steuerbar sind, Verwendung.

1.1 Stand der Technik

Künstliche neuronale Netze sind als adaptive Elemente in Regelungsaufgaben weit verbreitet [IWH95, WK00]. Wie die folgende Übersicht darstellt [SGL10], sind neben Bereichen wie Spracherkennung oder Bildverarbeitung auch sicherheitskritische Bereiche wie automotive oder medizinische Systeme vertreten, um nur einige zu nennen. Die Variationen künstlicher neuronaler Netze reicht vom bekannten Gradientenabstiegs-Lernverfahren, bis hin zur Modellierung und Simulation ganzer Hirnregionen auf molekularer Ebene [RA13]. Eine kleine Übersicht geeigneter Lernverfahren für Echtzeitanwendungen wird in [Ngu06] dargestellt.

Künstliche neuronale Netze (KNN) finden als adaptive Elemente auch Anwendung in der Luftfahrt, wobei sich zwei Anwendungs-Gruppen zur Sicherheitserhöhung besonders hervorheben:

- Model Reference Adaptive Control (MRAC)
- Nichtlineare Dynamische Inversion (NDI)

Während MRAC die Zustände eines Fluggerät-Modells mit den gemessenen Zuständen vergleicht, basiert die dynamische Inversion auf einem inversen Fluggerät-Modell. Kombinationen von MRAC und NDI sind allerdings auch denkbar [HSB11]. In den 2000er-Jahren fand im Bereich der NDI mit künstlichen neuronalen Netzen (KNN-NDI) zunehmends das sogenannte Pseudo-Control-Hedging (PCH) Beachtung, das die Berücksichtigung von Aktuatorbegrenzungen erlaubt und einen Beitrag zur Stabilität leistet [Joh00, JC00]. In [Cal96] wird eine KNN-NDI ohne PCH beschrieben. KNN-NDI mit PCH wird in [CHN00, Sim11, Lor10] auf einem Helikopter angewendet. [Now10, Hol04, Krü12] behandeln das Gebiet KNN-NDI mit PCH

[1] Norbert Wiener (∗1894 †1964) begründete mit dem Werk *Cybernetics or Control and Communication in the Animal and the Machine (1948)* die Wissenschaft der Kybernetik. Das Werk legt dar, wie technische Systeme, angelehnt an Prinzipien natürlicher Organismen, Informationen aufnehmen, verarbeiten und zur Regelung nutzen.

im Bereich unbemannter Flächenflugzeuge und [Joh00] für ein suborbitales Landegerät. [DAL13] stellt einen MRAC und [SCT15] ein KNN-NDI für Quadrocopter vor. Welche Reglerstruktur besser ist, kann noch nicht beantwortet werden, wie der Vergleich in [HSB11] darstellt. Der erste unbemannte Flugversuch mit einem kleinen Flächenflugzeug und KNN-NDI gelang erst im Jahr 2013 [CJC+13].

Während alle genannten Arbeiten auf dem Gradientenabstiegs-Lernverfahren basieren, gelingt [Krü12] die Erweiterung der KNN-NDI mit dem Sliding-Mode-Lernverfahren. Hierbei zeigt sich, dass dieses Lernverfahren deutlich robuster auf Störungen reagiert und zudem der Nachweis der Stabilität durch Betrachtung der KNN als zu regelndes System möglich ist.

Den meisten Arbeiten auf dem Gebiet der adaptiven Regelung mit unbemannten Flugzeugen, insbesondere der KNN-NDI ist gemein, dass die Reglerstrukturen in Simulationen getestet werden, d.h. eine Erprobung der Reglerstrukturen in realen Flugversuchen findet oft nicht statt. Dabei bieten sich gerade kleine unbemannte Flugzeuge als ideale Erprobungsplattform an. Die Gründe für seltene Flugversuche können nur erahnt werden. Eine Möglichkeit besteht darin, dass die Implementierung von Reglerstrukturen, die meist in digitalen Schaltplänen entwickelt werden, auf den embedded Computern mit C aufwendig ist. Dagegen zeigen Ergebnisse der NASA [RBB09], dass Autocode-Verfahren zur Programmierung von neuronalen Netzwerken eine effektive Alternative zum klassischen C darstellen. Gleichzeitig stellt sich die prinzipielle Frage der Implementierbarkeit der KNN-NDI auf kleinen Autopilotensystemen, die in [CJC+13, RBB09] nicht thematisiert wird.

1.2 Gegenstand der Arbeit

Die Einleitung verdeutlicht die Bedeutung von adaptiven Strukturen für die Sicherheit von unbemannten Flugzeugen. Die dynamische Inversion mit Sliding-Mode trainierten neuronalen Netzen wird dabei im Hinblick auf Stabilität besonders hervorgehoben. Gleichzeitig zeigt der Stand der Technik, die Vielzahl der Kombinationsmöglichkeiten in adaptiven Regelungssystemen und die seltene Durchführung von Flugversuchen, was die Frage nach einer Alternative zur C-Implementierung rechtfertigt.

Die vorliegende Arbeit leistet einen Beitrag zur praktischen Nutzung der nichtlinearen adaptiven dynamischen Inversion in unbemannten Flugzeugen. Dazu wird ein

Autopilotensystem vorgestellt, das die direkte Verwendung generischer Schaltpläne für Experimente im Bereich der Robotik erlaubt. Dies wird durch eine Autocode-Implementierungskette ermöglicht. Die adaptiven Regler in Form der KNN-NDI werden für das verwendete Flugzeug angepasst und erstmals mit dem Sliding-Mode-Lernverfahren auf einem sequentiellen Smartphone-Prozessor im Flugversuch getestet. Um das Potenzial der KNN-NDI zu untersuchen, werden verschiedene Reglerstrukturen gegeneinander, in Simulations- und Flugversuchen verglichen. Die Eigenschaften der Flugfähigkeits-Wiederherstellung mit der KNN-NDI wird zudem durch eine zusätzliche Ausfallsituation in allen Versuchen hervorgehoben. Zusätzlich wird mit dem Reference-Control-Hedging eine neuartige Variation des PCH vorgestellt, um einen Vergleich der dynamischen Inversion mit und ohne adaptive Elemente zu ermöglichen. Mit der Darstellung des Autopilotensystems inklusive der Implementierungskette und der Betrachtung des KNN-NDI Verhaltens in Bezug zum Prozessor wird die Verwendbarkeit von Autocode in Kombination mit Smartphone-Prozessoren für die KNN-NDI bestätigt.

1.3 Struktur der Arbeit

Die vorliegende Arbeit behandelt zwei Themengebiete, ein Autopilotensystem und die adaptive dynamische Inversion, die über Flugversuche und Hardware in the Loop Untersuchungen miteinander verknüpft werden.

Im Kapitel 2 wird dazu das Autopilotensystem vorgestellt. Die Schwerpunkte des Kapitels liegen in der Darstellung der Hardwarekomponenten, der Kommunikationsstruktur, dem Betriebssystem und der Implementierungskette. Das 3. Kapitel beschreibt den Aufbau der Simulation. Dazu wird die Flugzeugdynamik und die Elemente der Navigation und der Bahnregelung vorgestellt. Einleitend wird als Kontrast zu den Kapiteln der dynamischen Inversion, ein klassischer Linearregler beschrieben und ein erster Flugversuch mit dem Autopilotensystem, Navigation, Bahnführung und Linearregler im Schadensfall vorgestellt.

Kapitel 4 beschreibt die dynamische Inversion mit Pseudo-Control-Hedging, die Anpassung an die unbemannten Flugzeuge T200 bzw. Twinstar und das Reference-Control-Hedging. Hierbei werden Simulation und Flugversuch miteinander verglichen, um eine weitere Referenz zum späteren Flugverhalten mit adaptiven Elementen zu bilden. Anschließend werden in Kapitel 5 die künstlichen neuronalen Netze als adaptive Elemente vorgestellt und der Bezug zur dynamischen Inversion hergestellt.

Mit weiteren Anpassungen kann die adaptive dynamische Inversion im Kapitel 6 in Simulation und Flugversuch untersucht werden. Kapitel 6 analysiert zudem das Verhalten der verschiedenen Regler im Hardware in the Loop Verfahren in Bezug zum Autopilotensystem. Im letzten Kapitel werden die Erkenntnisse dieser Arbeit zusammengefasst und bilden einen Ausblick für zukünftige Forschungsansätze.

Flugversuche

Die Flugversuche finden mit der in Abbildung 1.1(a) dargestellten „Twinstar[2]“ der Firma Multiplex statt. Im Gegensatz dazu sind die Parameter der Flugzeugdynamik mit dem unbemannten Flugsystem „Carolo T200“ der TU Braunschweig verfasst, vgl. Abbildung 1.1(b). Aus diesem Umstand resultieren Modellfehler in der nichtlinearen Regelung. Den Argumentationen von [Krü12, Hol04] folgend, können Modellfehler allerdings zur Bestätigung der adaptiven Eigenschaften der zu untersuchenden Strukturen gewertet werden. Die wichtigsten technischen Daten beider Fluggeräte werden in Tabelle 1.1 dargestellt.

(a) Multiplex Twinstar.

(b) Carolo T200.

Abbildung 1.1.: Darstellung der unbemannten Flugzeuge Twinstar und T200.

	Multiplex Twinstar	**Carolo T200**
Startgewicht	1 kg	5 kg
Spannweite	142 cm	200 cm
Länge	109 cm	120 cm
Reisefluggeschwindigkeit	15 m/s	20 m/s
Antrieb	elektrisch	elektrisch

Tabelle 1.1.: Technische Daten der unbemannten Flugzeuge Twinstar und T200

[2]Die genaue Modellbezeichnung des Herstellers Multiplex lautet BK Twinstar II

2 UAV-Autopilotensystem

Obwohl viele Veröffentlichungen im Bereich der Algorithmen von UAV-Autopiloten existieren, wie z.B. Navigationsalgorithmen, Bahnführung oder Regelung, sind Publikationen, die den Autopiloten als Gesamtsystem betrachten, selten [Fer09, Kit12]. Bei den Veröffentlichungen zu den Algorithmen wird oft außer Acht gelassen, dass die Wahl der Autopilotenhardware, das Betriebssystem und die Implementierungskette eine nicht zu vernachlässigende Synergie mit den Algorithmen selbst bildet.

Sensor- und Aktuatorfehler haben einen Einfluss auf die Mess- und Regelgüte der Flugexperimente:

Da Mess- und Stellverzögerungen, Sensorfehler und Aktuatorfehler einen Einfluss auf die Güte einer zu implementieren Reglerarchitektur haben können, müssen diese so weit wie möglich bekannt sein und sich mindestens auf einen vernachlässigbar kleinen Zeit- und Fehlerrahmen beschränken.

Das Betriebssystem und die Prozessorkommunikation führen zu Verzögerungen:

Neben Verzögerungen, die durch Kommunikationen der einzelnen Peripheriebausteine entstehen, ist die Wahl des/der Betriebssystem/e sowie der Kommunikationsprotokolle betrachtenswert. Wie bei Sensoren und Aktuatoren müssen diese Verzögerungen so weit wie möglich bekannt sein.

Gewicht und Stromverbrauch der Hardware beschränken die Anwendbarkeit:

Wenngleich ein moderner Desktop-Prozessor enorme Rechenkapazitäten besitzt, ist er in kleinen UAV kaum einsetzbar. Das Gewicht und der Stromverbrauch der Autopilotenhardware beschränken somit die verfügbaren Rechenressourcen.

Die Komplexität der zu untersuchenden Algorithmen bestimmt die Prozessorwahl und die Implementierungskette:

Die Verwendung von adaptiven Elementen stellt a priori nicht kalkulierbare Anforderungen an die Prozessor-Kapazitäten. Um den Zusammenhang zwischen Algorithmus und Kapazität herzustellen, wird der gewählte Prozessor und die aus den

Algorithmen resultierende Auslastung des Prozessors im Laufe der Arbeit dargestellt. Auch die Wahl der Implementierungswerkzeuge hat eine indirekt beschränkende Wirkung auf die Komplexität der zu untersuchenden Algorithmen. So können neuronale Netzwerke auch in analoger Hardware nachgebildet werden, allerdings ist der Zeitaufwand der Implementierung enorm [Dra00]. Die Automatisierung der Implementierungskette trägt zur Realisierbarkeit des Projektes bei.

2.1 Aufbau des Autopiloten

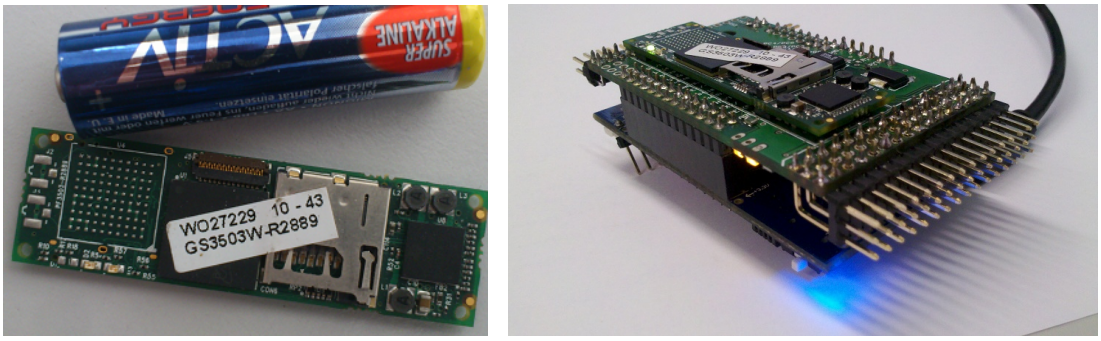
Um den hohen Ressourcen-Anforderungen der angestrebten Regler- und Navigationsarchitektur gerecht zu werden, wurde eigens dafür ein, in der folgenden Kombination, neuartiges Autopilotenkonzept entwickelt. Die Verwendung von neuronalen Netzwerken wird oft mit der Voraussetzung eines parallel arbeitenden Prozessors, z.B. FPGA (Field-Programmable Gate Array) in Verbindung gebracht [OR06, CJC+13, CPN06]. Dabei ist die Portabilität nicht immer garantiert und Algorithmen müssen ggf. spezifisch für den jeweiligen FPGA in VHDL (Very High Speed Integrated Circuit Hardware Description Language) codiert werden [CHU06]. Dagegen haben klassische, sequentiell arbeitende Prozessoren einen wesentlichen Vorteil: Algorithmen werden unabhängig vom Prozessor in einer geeigneten Sprache, z.B. C oder Matlab entwickelt und sind zu allen in der Compiler Bibliothek implementierten Prozessoren kompatibel. Um noch einen Schritt weiter zu gehen, können diese inzwischen mit einem sogenanntem „Autocode“ Generator aus einem, auf Piktogrammen und Symbolen basierten Schaltplan einer Simulationsumgebung, direkt mit diesem Schaltplan programmiert werden. D.h. der Regelungstechniker wird vom Text-basierten Programmierer zum Designer, der nur noch mit dem Schaltplan arbeitet. Diese Technik erlaubt die Programmierung von hochkomplexen Schaltplänen mit möglichst geringen Programmierer-Ressourcen.

Um den Gewichtsanforderungen gerecht zu werden, gerade im Bereich der unbemannten Kleinstflugzeuge, werden kleine stromsparende Prozessoren benötigt. Gleichzeitig führt die Entwicklung der letzten Jahre in der Smartphone- bzw. Tabletindustrie, zu immer performanteren ARM- (Advanced Reduced Instruction Set Computers) Prozessoren, die im Gegensatz zur klassischen Intel- oder AMD Desktop-Prozessorarchitektur nur einen Bruchteil der Energie benötigen und dennoch hohe Taktraten aufweisen. Während zu Beginn dieses Projektes (Jahr 2010) noch ein Smartphone-Prozessor mit 800 MHz Taktrate zur Verfügung stand, sind bereits bei

Abschluss des Projektes (Jahr 2014) ARM-Prozessoren mit vier Kernen und einem Basistakt von 2,5 GHz in Smartphones üblich.

Die Grundidee des hier vorgestellten Autopilotenkonzeptes basiert auf der Kombination der Modularität eines „kleinen“ ARM-3 Datenakquise-Prozessors und den Möglichkeiten eines „großen“ ARM-8 Rechenkerns. Gerade die praktischen Anwendungen von unbemannten Flugzeugen haben gezeigt, dass die Einbindung von mehreren Mess- oder Aktuator-Modulen deutliche Vorteile bieten kann [KWV11, MB11, KB12]. Als kleiner Prozessor wurde dazu der STM32-F103 von STMicroelectronics ausgewählt, um mit seinem vielfältigen Peripherieangebot die Verbindung zu diversen Sensoren und Aktuatoren zu sichern. So besteht mit einem CAN-Bus im STM32-Prozessor die zusätzliche Möglichkeit, die Verbindung zu weiteren Modulen herzustellen [KK14].

Als „großer“ Prozessor kam nur der Texas Instruments OMAP3530 in Frage, da dieser alternativlos (im Jahr 2010), als leistungsfähigster Prozessor auf dem kleinsten Mainboard „Gumstix“ zur Verfügung stand. Die Gumstix, wie in Abbildung 2.1(a) dargestellt, sind mit 58 mm x 17 mm kaum größer als eine handelsübliche AA-Batterie und werden auf den restlichen Autopiloten aufgesteckt, wie in Abbildung 2.1(b) ersichtlich. Mit Sensorik und GPS wiegt der gesamte Autopilot 60 Gramm bei den Abmessungen von 7 cm x 4 cm und einer Höhe von 3 cm. Trotz der hohen Taktrate von 800 MHz kommt der gesamte Autopilot auf einen Stromverbrauch von maximal 2,5 Watt.



(a) Gumstix Computer mit OMAP 3530. (b) Autopilot mit montierten Gumstix.

Abbildung 2.1.: Der „große“ Rechencomputer Gumstix wird auf den restlichen Autopiloten aufgesteckt.

In der Grafik 2.2 wird die gesamte Kommunikationsstruktur innerhalb des Autopiloten ersichtlich, deutlich zu erkennen ist der „kleine“ STM32 Prozessor sowie der

„große“ OMAP 3530 Rechenkern. Die beiden Prozessoren sind untereinander mit dem sogenannten SPI (Serial Peripheral Interface) verbunden. Diese Verbindung erlaubt eine bidirektionale Übertragungsrate von 2 MHz und ist damit die schnellste zur Verfügung stehende echtzeitfähige Schnittstelle, um die anfallenden Sensor- und Aktuatordaten zwischen „kleinem“ und „großem“ Prozessor zu kommunizieren. Weiterhin werden die Sensordaten der Gyroskope, Beschleunigungssensoren sowie

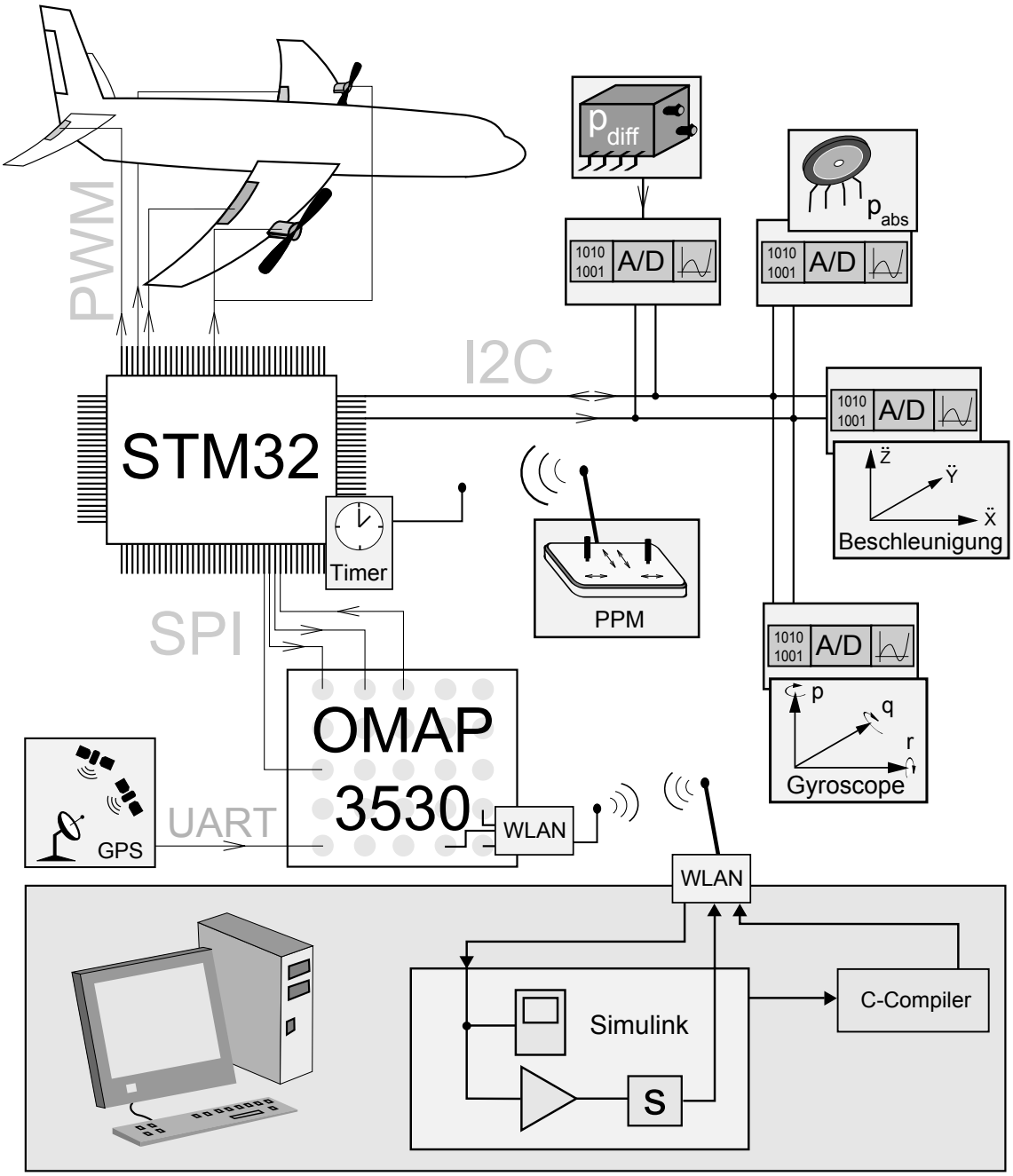


Abbildung 2.2.: Struktur des Autopilotensystems.

Drucksensoren über die sogenannte „I2C“ Schnittstelle vom STM32 Prozessor angefordert. Die GPS-Daten werden über eine serielle UART-Schnittstelle übermittelt. Die Steuerung des Systems erfolgt über eine, im Modellbau übliche Funkfernbedienung, deren PPM (Puls Pause Modulation) Signale von einem Zeitmesser innerhalb des STM32 Prozessors ausgewertet werden. Je nach Bedarf wird die Aktuatorik des Flugzeugs von den Funkfernbedienungssignalen oder im Bahnreglermodus direkt vom Autopiloten gesteuert. Eine Übersicht der Schnittstellentypen wird in [Kuf09] dargestellt.

2.2 Inertialsensorik

Als Inertialsensorik kommt die IMU3000 Combo (vgl. Abbildung 2.3) zum Einsatz, deren Sensoren üblicherweise in mobilen Geräten verbaut wird. Diese MEMS (Micro Electro Mechanical Systems) IMU ist eine Kombination aus dem Drei-Achs-Gyroskop IMU3000 und dem Drei-Achs-Beschleunigungssensor ADXL345. Die Gy-

Abbildung 2.3.: IMU3000 Combo, mit IMU3000 Gyroskop und ADXL345 Beschleunigungssensor.

roskope basieren auf vibrierenden Piezo Stäbchen, die durch Kombination mit dem Coriolis-Effekt, eine zur Drehrate proportionale Spannung erzeugen. Diese Spannung wird vom chipinternen A/D Wandler gemessen und mit einem Takt von 100Hz vom STM-32 Prozessor ausgelesen. Der Messbereich der Gyroskope liegt bei $\pm 2000°/s$ bei einer Auflösung von 16 Bit. Der Beschleunigungssensor hingegen basiert auf der Messung des Abstandes einer Masse gegenüber einem Referenzrahmen. Der Messbereich der Beschleunigungssensoren liegt bei ± 8 g mit einer Auflösung von 10 Bit. Eine weitergehende Beschreibung der Funktionsweisen von MEMS-Sensorik kann [Kuf09, Win06] entnommen werden.

Bevor die IMU eingesetzt werden kann, muss diese kalibriert werden. Neben Skalierung und Offset hat sich in Untersuchungen gezeigt, dass die Sensortemperatur einen großen Anteil am Sensorausgang der IMU hat. Zu diesem Zweck wurde ein am Institut vorhandener Drehtisch mit einer darauf montierten, geregelten Temperaturkammer verwendet. Weiterhin ist bereits in der IMU3000 ein vom Hersteller verbauter Temperatursensor vorhanden. Dies erlaubte die Untersuchung der Temperatureinflüsse im Bereich von -5 °C bis +30 °C. Es zeigt sich, dass in den verwendeten Gyroskopen ein hoher Anteil des Offsets von der Temperatur abhängig ist. Ein Temperaturoffset ist auch in den Beschleunigungssensoren sichtbar, der Einfluss von 0.1 LSB/°C wird aber als vernachlässigbar gering angesehen. Temperaturabhängige Skalierungsänderungen konnten in den Sensoren nicht festgestellt werden. Zur Kalibrierung der IMU wird folgendes Sensor-Modell der gemessenen Drehraten-Rohdaten $\vec{\omega}_{raw} = [p_{raw}, q_{raw}, r_{raw}]^T$, in Abhängigkeit der tatsächlichen Drehraten $\vec{\omega} = [p, q, r]^T$ herangezogen:

$$\vec{\omega}_{raw} = \vec{S}_{\omega} \cdot \vec{\omega} + \vec{\omega}_{offset} + \vec{T}_{\omega,offset} \cdot T_{IMU}. \tag{2.1}$$

Dabei entspricht $\vec{\omega}_{offset}$ dem statischen Gyroskop Offset, T_{IMU} der Sensortemperatur, und $\vec{T}_{\omega,offset}$ dem temperaturabhängigen Offset sowie $\vec{S}_{\omega}$ den Skalenfaktoren. Die kalibrierten Messdaten können einfach durch Umstellen nach $\vec{\omega}$ berechnet werden. Das Sensormodell für die Beschleunigungssensoren ergibt sich analog, allerdings ohne Temperaturkompensation.

$$\vec{a}_{raw} = \vec{S}_{a} \cdot \vec{a} + \vec{a}_{offset}. \tag{2.2}$$

Der Vollständigkeit halber soll erwähnt werden, dass neben Skalierungsfaktoren, Offsets und Temperatureinfluss auch weitere unberücksichtigte Sensorfehler einen Einfluss auf die Messung haben können: Diese sind Nichtlinearitäten, Skalenfaktorasymmetrien, Totzonen und Quantisierungsfehler. In [Win06] werden Fehlausrichtungen der einzelnen Sensoren untereinander kalibriert. Daneben zeigen jüngste Untersuchungen einen störenden Einfluss von Beschleunigungskräften auf die Drehratensensoren, die sogenannte G-Sensitivität [Ana11, BL12].

Das Gyroskop ist mit einem internen Hochpass ausgestattet, dessen Grenzfrequenz zu 98 Hz, unter Berücksichtigung des Regeltaktes von 100 Hz, gewählt wurde. Trotz des Hochpass-Filters kommt es, neben dem werksseitig angegebenen weißen Rauschen, zu unerwünschten, vom Motor induzierten Rauschtermen, wie in Abbildung 2.4 beispielhaft an der Gierrate dargestellt. Es liegt nahe, die zwei Amplituden bei

36,5 Hz und 38,5 Hz als Drehzahlen der zwei verwendeten Motoren zu interpretieren. Die Amplitude dieses Motorrauschens hängt dabei mit der Entfernung zwischen Motor und Sensor zusammen, wobei die Motorschwingungen über das Flugzeug übertragen werden. Elektromagnetische Effekte können ausgeschlossen werden, da durch mechanische Entkopplung das Motorrauschen verschwindet. Ob hierbei möglicherweise die G-Sensitivität einen Rolle spielt, bleibt zu untersuchen.

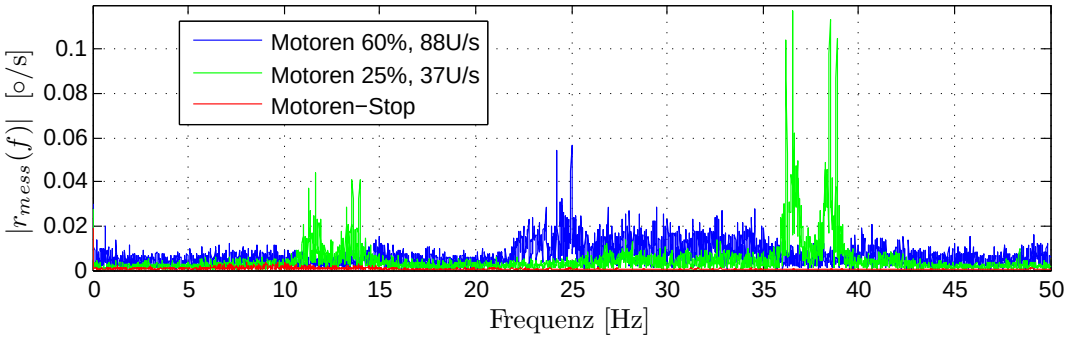


Abbildung 2.4.: Einseitiges Spektrum der Gierrate bei verschiedenen Motordrehzahlen.

Wie zu erwarten, zeichnet sich in den Beschleunigungssensordaten ein ähnliches Bild ab, wie in Abbildung 2.5 dargestellt.

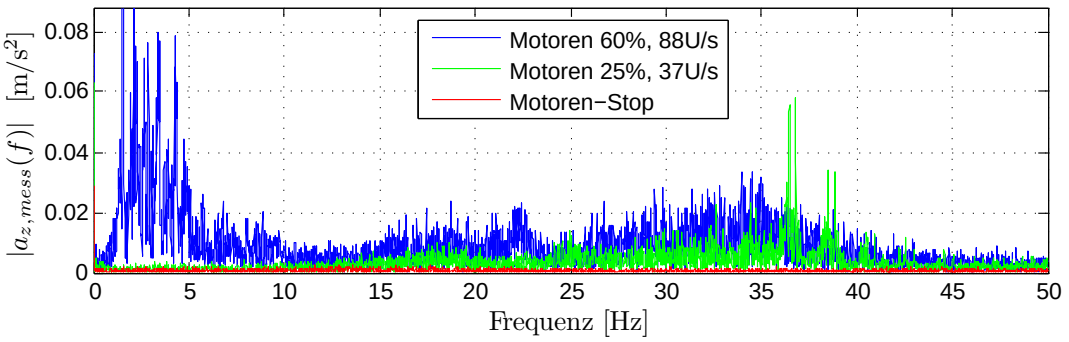


Abbildung 2.5.: Einseitiges Spektrum des Z-Beschleunigungssensors bei verschiedenen Motordrehzahlen.

Bereits in [Win06] wurden im Flugversuch Genauigkeitsunterschiede zwischen ruhender IMU und der IMU im Flugversuch festgestellt, die nicht zugeordnet werden konnten. Es ist nicht auszuschließen, dass dieser Fehler auf das hier dargestellte Phänomen zurückzuführen ist.

Um das Motorrauschen in der späteren Simulation zu berücksichtigen, wird das Sensormodell aus Gleichung (2.1) bzw. (2.2) mit den zu erwartenden Störungen erweitert. Dazu wird neben weißem Rauschen u_{noise}, ein Motorrauschterm u_{motor} hinzugefügt. Für das Drehraten-Sensormodell gilt demnach,

$$\vec{\omega}_{raw} = \vec{\omega} + \vec{\omega}_{offset} + \vec{T}_{\omega,offset} \cdot T_{IMU} + \vec{K}_{\omega,n} \cdot u_{noise} + \vec{K}_{\omega,m} \cdot u_{motor}, \tag{2.3}$$

mit den Verstärkungsfaktoren $\vec{K}_{\omega,m}$ und $\vec{K}_{\omega,n}$ der jeweiligen Achsen. Diese Verstärkungsfaktoren müssen je nach Flugzeugmodell neu angepasst werden. Analog gilt für das Beschleunigungs-Sensormodell:

$$\vec{a}_{raw} = \vec{a} + \vec{a}_{offset} + \vec{K}_{a,n} \cdot u_{noise} + \vec{K}_{a,m} \cdot u_{motor}. \tag{2.4}$$

Der Motorrauschterm u_{motor} bildet sich aus einer Drehzahl f_{rpm} der zwei Motoren. Diese Drehzahl ist reglerbedingt leichten Schwankungen unterworfen, die mit der Frequenz f_1 abgebildet werden.

$$u_{motor} = sin((sin(f_1 \cdot t) + f_{rpm}) \cdot t). \tag{2.5}$$

2.3 Drucksensoren

Um im Flug die echte Fluggeschwindigkeit messen zu können, wird ein Pitotrohr in Kombination mit dem Differenzdrucksensor MPXV5004DP von Freescale eingesetzt. Dieser Sensor arbeitet in einem Messbereich von 0 bis 3,92 kPa und liefert als Messergebnis eine zum Druck proportionale Spannung. Diese Messspannung wird von einer Operationsverstärkerschaltung skaliert und mit einer Grenzfrequenz von 50 Hz tiefpassgefiltert (vgl. Skalierung, Offsetabgleich und Filtertechniken mit Operationsverstärkern [Man01]). Das gefilterte, analoge Signal wird anschließend mit einem A/D-Wandler in ein digitales Signal gewandelt. Zur Kalibrierung der Pitotsonde eignet sich der Vergleich zwischen GPS-Geschwindigkeit und Messdruck, z.B. aus Fahrversuchen an windstillen Tagen.

Zur Stützung der Höhenmessung kommt der Absolutdrucksensor MPXA6115 von Freescale zum Einsatz. Mit einem Messbereich von 15 bis 115 kPa ist damit nach DIN 5450 eine Höhenmessung bis 13,5 km über Meereshöhe möglich. Wie beim Differenzdrucksensor, ist eine Anpassung der analogen Signale mit Operationsverstärkern nötig. Die Kalibrierung der Höhendrucksensor-Skalierung erfolgt über einen Abgleich mit Höhendaten des GPS-Empfängers aus einem Flugversuch. Da der Höhen-Offset

des Absolutdrucks wetterabhängig ist, bietet sich zur Höhenmessung eine Komplementärfilterung der Absolutdruck- und GPS-Höhensignale an.

2.4 GPS-Empfänger

Im Rahmen dieser Arbeit kommt der LEA-6H GPS-Empfänger der Firma u-blox zum Einsatz. Mit einer Frequenz von 4 Hz wird mit dem GPS-Empfänger Höhe und Position des unbemannten Fluggerätes bestimmt. Die Genauigkeit des Empfängers liegt laut Datenblatt [u13] in einem Radius von 2,5 m um den Empfänger.

2.5 Aktuatoren

In der später dargestellten dynamischen Inversion haben Verzögerungen und Totzeiten der Aktuatoren einen wesentlichen Einfluss auf die Regelgüte. Vor allem die Servomotoren der Quer- und Höhenruder beeinflussen die Reaktionen des Flugzeugs maßgeblich. Zur realistischen Abbildung der Flugzeugsimulation und für die Umsetzung der dynamischen Inversion wird somit die Modellierung der Aktuatorik benötigt. Erste Ergebnisse zur Aktuatorvermessung finden sich in [Sch02]. Zur Berücksichtigung von Latenzen im hier verwendeten Autopiloten, wurden die verwendeten Aktuatoren zudem von [Sch13] im Hardware in the Loop (HIL) Verfahren untersucht. Die Ergebnisse der Aktuatoruntersuchungen führen zu einer Modellierung der Aktuatoren als System zweiter Ordnung mit einer Totzeit. Erste Untersuchungen zeigen zudem, dass über ein Messprinzip mit Hallsensoren die Messung der Ruderstellung im geschlossenen Kreis auch für einen Flugversuch geeignet ist [Kra13].

Ein nicht gelöstes Problem stellt die Kommunikation mit den Servos dar. Die Ansteuerung von Standard-Modellbau-Servos erfolgt, je nach Servo, asynchron alle 20 ms oder 10 ms. Ein Ansatz zur Synchronisation der Servo-Steuerung mit dem Reglertakt von 100 Hz existiert bis jetzt noch nicht. Denkbar sind z.B. über I2C ansteuerbare Servos, um höhere Kommunikationsfrequenzen zu erlauben.

2.6 Echtzeitbetriebssystem

Die Verwendung eines leistungsfähigen Prozessors hat einen weiteren Vorteil: Die Möglichkeit der Verwendung von Standard-Linux als Echtzeit-Betriebssystem mit

seinen zahllosen Bibliotheken, Hardwaretreibern und Diagnosetools. Zwar gibt es zahlreiche Anbieter von auf Linux basierenden Echtzeitbetriebssystemen, als Beispiel sei an dieser Stelle „QNX“ genannt, diese Alternativen reichen aber an den Fundus der Möglichkeiten des Standard-Linux nicht heran. Zum Zeitpunkt der Recherche (Jahr 2010) waren z.B. keine SPI-, WLAN- oder USB-Treiber für den OMAP-3530 Prozessor in QNX verfügbar. Ein weiterer Aspekt ist die mögliche Zertifizierbarkeit des echtzeitfähigen Standard-Linux-Kernels, die in [Kam11] ausführlich diskutiert wird. Die Verfügbarkeit einer WLAN-Verbindung erlaubt die drahtlose Programmierung und die drahtlose Wartung über den sogenannten External-Mode in Simulink. Das bedeutet, die in Echtzeit generierten Daten können im zugehörigen Simulink-Modell auf einem externen PC betrachtet oder aufgezeichnet werden, was sich besonders in HIL- (Hardware in the Loop) Experimenten als hilfreich erwiesen hat.

Klassische Betriebssysteme wie Windows, Linux oder OS X sind für Echtzeitanwendungen nicht geeignet. Ein Betriebssystem organisiert unter anderem die Ausführungen verschiedener Aufgaben, sogenannter Prozesse. Die Organisation ist darauf ausgerichtet, dem Benutzer ein möglichst „flüssiges“ Arbeitserlebnis zu bieten. So ist es möglich, eine Video-Datei abzuspielen und gleichzeitig ein Textdokument zu bearbeiten. Der Organisator des Betriebssystems, der Scheduler, gibt dabei den Prozessen abwechselnd die Möglichkeit, den Prozessor zu nutzen [Tan02]. Die Problematik geht einen Schritt weiter, wenn Peripheriegeräte, z.B. die Maus, ein Signal an das Betriebssystem senden, also einen sogenannten Hardware-Interrupt generieren. In diesem Moment werden kurzzeitig alle anderen Prozesse unterbrochen und die Daten der Maus werden in einer Interruptroutine ausgewertet, während alle anderen Prozesse warten. Diese Interruptroutine ist so kurz, dass der Benutzer davon nichts merkt. Allerdings würden zeitkritische Prozesse an dieser Stelle um wertvolle Millisekunden verzögert. Einen praktischen Einstieg in das Scheduling von Prozessen und embedded Echtzeit-Linux bietet [Ang09].

Ohne zu tief in die Thematik einzusteigen, kann das Prinzip des Scheduling in Linux beginnend mit Abbildung 2.6, angelehnt an [Ang09], dargestellt werden. Ein Echtzeit-Prozess soll zu gewünschten Zeitpunkten aufgerufen werden: Der Scheduler reiht diesen Prozess neben einen Kernelprozess in eine Abarbeitungsliste ein und startet diesen. Kurz darauf wird der Kernelprozess gestartet, z.B. durch Aufrufen des Video-Treibers. Erst nach der Beendigung des Kernelprozesses wird wieder der Echtzeit-Prozess aufgerufen, diesmal mit einer Verzögerung. Ein ähnliches Szenario spielt sich ab, wenn ein anderer Usermode-Prozess, z.B. die erwähnte Textverarbei-

tung, aufgerufen wird. Erst beim nächsten Aufruf des Schedulers wird der Echtzeit-Prozess abgearbeitet. Die Granularität des Schedulers, beim Standard Linux-Kernel sind dies 4 ms, führt zu einer zusätzlichen Verzögerung. Letztendlich führt der erwähnte Interrupt ebenfalls zur Unterbrechung des Prozesses.

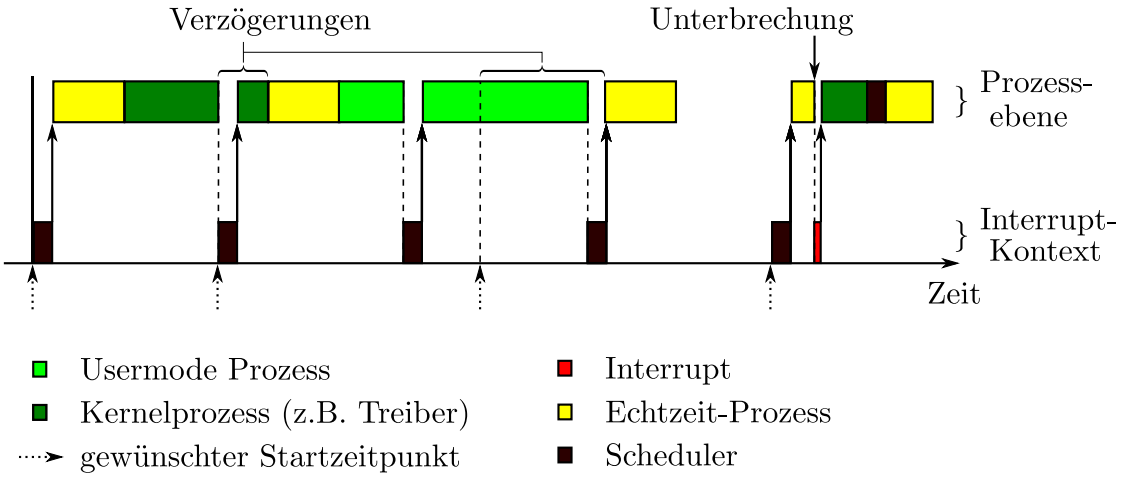


Abbildung 2.6.: OMAP-3530, ohne Echtzeitpriorität, (vgl. [Ang09]).

Mit dem sogenannten „Cyclictest“ [Bau07] kann die vergangene Zeit zwischen dem Prozessaufruf und der tatsächlichen Ausführungszeit anschaulich illustriert werden. Abbildung 2.7 stellt dazu die auf dem OMAP-3530 Prozessor gemessene Verzögerung ohne Echtzeitmodus dar. Während sich die Verzögerung ohne andere Prozesse auf eine Obergrenze von ca. 40 μs begrenzt, wird die Verzögerung (Abbildung 2.7, roter Bereich) durch andere Prozesse unberechenbar erhöht.

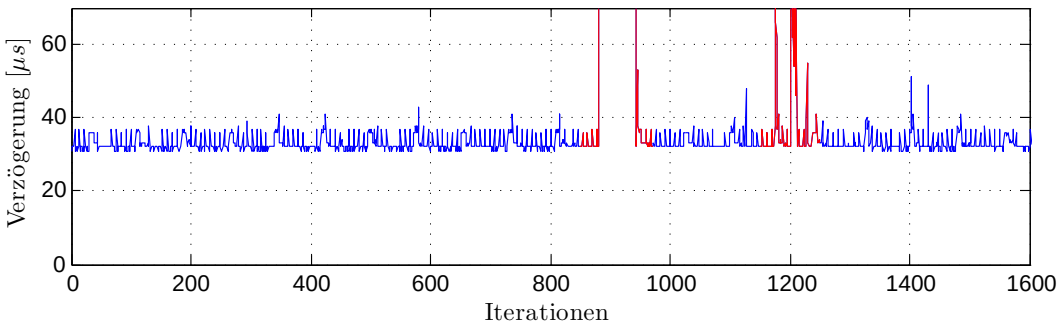


Abbildung 2.7.: OMAP-3530, Cyclictest ohne Echtzeitpriorität, unvorhersehbare Latenzen (rot) durch andere Prozesse und Interrupts.

Um eine zeitkritische Anwendung auszuführen ist es daher sinnvoll, diese Anwendung mit der höchsten Priorität im Betriebssystem auszustatten, um zu garantieren, dass keine andere Anwendung oder ein Interrupt den dringend benötigten Prozessor verwendet. Windows, Linux und andere Betriebssysteme ohne Echtzeit scheiden an dieser Stelle aus. Ein nicht unterbrechbarer Prozess ist dort unerwünscht, dieser würde das „flüssige“ Arbeitserlebnis stören.

Der Preempt-RT Patch bietet eine Lösung für den Standard-Linux Kernel, um die unerwünschten Latenzen zu verhindern. Der Kernelcode wird mit diesem Patch zum voll preemptiven Kernel modifiziert und die Option CONFIG_PREEMPT_RT wird in der Kernelkonfiguration verfügbar. Dazu werden durch insbesondere drei Mechanismen die Verzögerungen auf ein berechenbaren Maß herabgesetzt:

Priorität
Ohne Preempt-RT Patch haben alle Prozesse dieselbe Priorität, das bedeutet, die Prozesse sind für den Scheduler gleichwertig und die Abarbeitung dieser Prozesse folgt keiner für Echtzeitanwendungen nützlichen Ordnung. Mit der Kernelversion 2.6 wurde die Option CONFIG_PREEMPT in der Kernelkonfiguration eingeführt, die es erlaubt, den Prozessen eine Priorität zuzuweisen. Allerdings können immer noch Spinlocks[1] verhindern, dass ein Prozess mit höchster Priorität den Kernelprozess unterbrechen kann. Der Preempt-RT Patch ersetzt die Spinlocks durch Mutexe[2], womit ein Echtzeitprozess die höchste Autorität im gesamten Betriebssystem erreicht, also auch den Kernelprozess unterbrechen kann. Warum diese Option nicht in Standard-Kernel verfügbar ist, wird an dieser Stelle ersichtlich: Ein Programmierfehler, z.B. eine endlose Schleife, führt zwangsläufig zu einem Dead-Lock, d.h. das Betriebssystem reagiert nicht mehr und kann nur durch einen Neustart wieder in Betrieb genommen werden.

HR-Timer
Die bereits erwähnte Granularität des Schedulers von 4 ms stellt für hochfrequente Echtzeitanwendungen eine weitere Hürde dar: Die hier zu implementierende Reglerstruktur soll mit 100Hz arbeiten, die Ausführung einer Regleriteration wäre aber erst nach dem Aufruf des Schedulers möglich. Dazu wurde mit dem Preempt-RT Patch ein vom Scheduler unabhängiger Timer, der sogenannte HR-Timer (High-Resolution Timer) eingeführt. Wie in Abbildung 2.8 dargestellt, können die Echtzeitprozesse unabhängig vom Scheduler ausgeführt werden.

[1] Spinlocks sind Blockaden, die durch Warten verhindern, dass ein anderer Prozess in den durch diese Blockade geschützten Bereich eintreten darf.
[2] Mutexe sind Marker, die verhindern, dass ein anderer Prozess auf eine Ressource zugreifen darf.

Software-Interrupts

Ein Hardwareinterrupt ist ein fester Bestandteil des Prozessors und kann, sofern er nicht abgeschaltet wird, nicht verhindert werden. Die Interruptroutine, also z.B. das Verarbeiten der Maus-Daten, ist aber mit dem Preempt-RT Patch in einem eigenen Prozess, mit einer eigenen Priorität implementiert. Die Priorität des Echtzeitprozesses kann also über die Priorität der Interruptroutine gestellt werden. Wie in Grafik 2.8 dargestellt, unterbricht der Hardware-Interrupt zwar den Echtzeitprozess, der zugehörige Kernel-Treiber wird aber erst nach dem Echtzeitprozess ausgeführt.

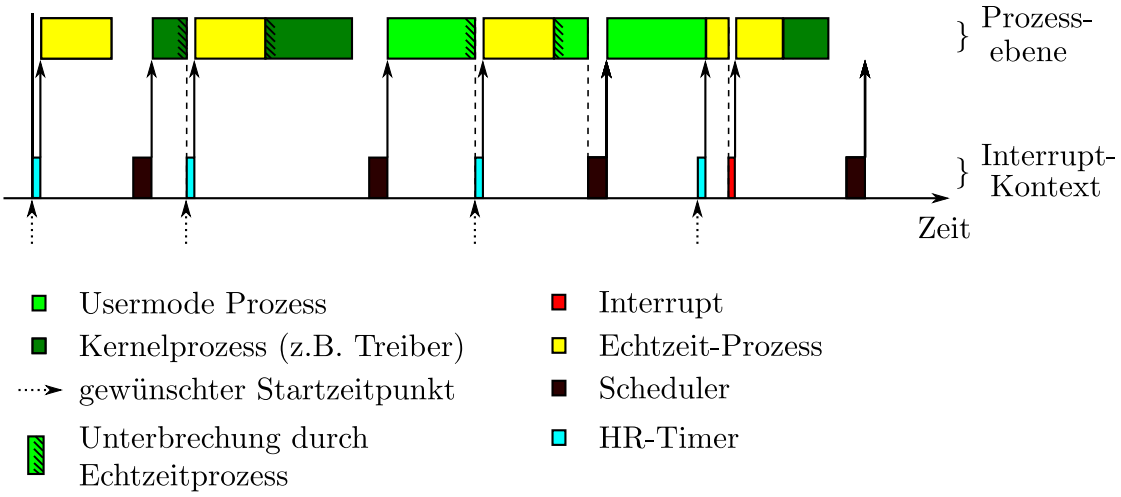


Abbildung 2.8.: OMAP-3530, mit Echtzeitpriorität, (vgl. [Ang09]).

Wird der Echtzeittest mit dem Preempt-RT Patch wiederholt (vgl. Abbildung 2.7 und 2.9), können verschiedene Programme und Interrupts nebenläufig ausgeführt werden und die Ausführungslatenz bleibt auf dem Linux des OMAP-3530 Prozessor unter 65 μs. In Abbildung 2.9 sind die Bereiche, in denen andere Programme und Interrupts arbeiten, deutlich zu sehen. In diesen Bereichen steigt die durchschnittliche Latenz von 21 μs auf 30 μs an. Derartige Tests werden zudem als Langzeittests von OSADL (Open Source Automation Development Lab) mit verschiedenen Prozessoren durchgeführt, unter anderem mit der OMAP35xx Prozessorfamilie, um Fehlerquellen auszuschließen und die Langzeitstabilität der Latenzen zu demonstrieren [OSA14].

2.7 Datenlogger

Die Iterationskette zwischen Simulationen und Flugversuchen fordert die Möglichkeit, möglichst viele Flugdaten speichern zu können. Mit einer Wiederholrate von

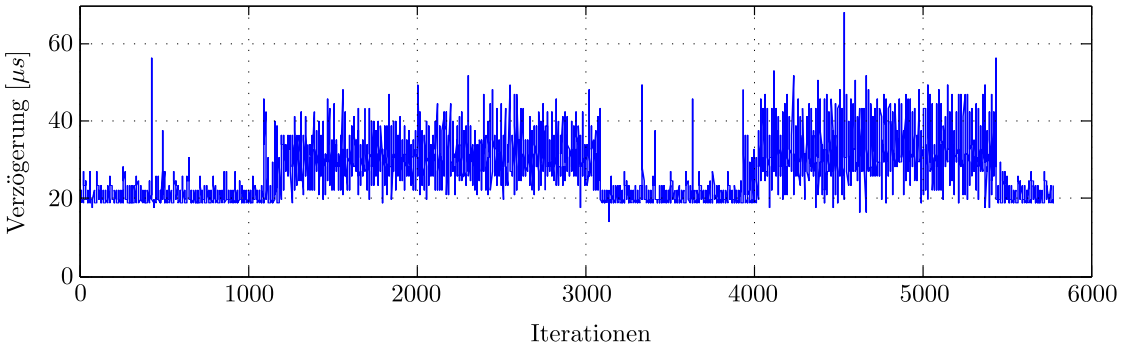


Abbildung 2.9.: Cyclictest OMAP-3530: Höchste Echtzeitpriorität, Latenzen auch bei 100% CPU Auslastung unter 65 μs begrenzt.

100 Hz und einem Datenvektor aus 64 Bit Fließkomma-Signalen können beachtliche Datenmengen entstehen. Im Rahmen dieser Arbeit wird ein Datenvektor von 60 Signalen zur Wiedergabe der flugzeuginternen Zustände abgespeichert. Dies führt zu einer Datenrate von 2 MByte/min. Eine direkte, echtzeitfähige Speicherung dieser Daten auf die Speicherkarte des Gumstix-Computers ist nicht möglich, da andere Prozesse und das Betriebssystem die Speicherkarte blockieren können. Daher wird ein eigener, nicht echtzeitfähiger Prozess zur Speicherung der Messdaten verwendet. Dieser Prozess puffert die Messdaten im RAM des OMAP-3530 Prozessors und schreibt diesen Puffer kontinuierlich auf die Speicherkarte.

2.8 Prozessor-Kommunikation

Betrachtet man im weiteren Verlauf der Arbeit den Aufbau der jeweiligen Reglerkaskaden, so wird deutlich, dass die hochfrequente Messung der Drehraten und Beschleunigungsdaten möglichst latenzfrei ablaufen muss. Andernfalls besteht die Gefahr von Phasenverschiebungen innerhalb der Reglerstruktur. Gleichzeitig muss sichergestellt werden, dass die Beschleunigungs- und Drehratenmessung für die Positions- und Lagebestimmung im Navigationsfilter minimale Latenzen aufweist [Win06]. Die Prozessorkommunikation hat damit einen direkten Einfluss auf die Verzögerungen der gesamten Reglerarchitektur. In Anbetracht der relativ langsamen Höhenregler- und Geschwindigkeitsregler-Dynamik wird auf eine weitere Betrachtung der Latenzen in der barometrischen Absolut- und Differenzdruckmessung verzichtet.

In Abbildung 2.10 ist die Kommunikationsstruktur der zeitkritischen Komponenten dargestellt. Mit dem OMAP-3530 Prozessor als Master wird die gesamte Synchronisation der Struktur eingeleitet. Der HR-Timer des OMAP-3530 startet dabei alle

10 ms einen Reglertakt, so dass eine Reglerfrequenz von 100 Hz erreicht wird. Der Prozess[1] des Reglertaktes startet allerdings mit der bereits erwähnten Echtzeitverzögerung von $< 65\ \mu$ s. Bevor der Regleralgorithmus (inkl. Navigation, Bahnführung, neuronale Netze, etc.) startet, werden über den SPI-Port alle Signale, z.B. Sensordaten, mit dem STM-32 Prozessor kommuniziert. Dazu werden die Signale, z.B. die Sensordaten, mit einem Kommunikationsprotokoll zu einem Puffer zusammengefasst und nach erfolgter Übertragung wieder in Signale zerlegt. Deutlich zu sehen ist an Grafik 2.10, dass neben der Datenübertragung selbst, auch das Kommunikationsprotokoll Zeit in Anspruch nimmt. Bei 72 MHz dauert diese Signalverarbeitung auf dem STM32-Prozessor deutlich länger, als auf dem schnellen OMAP-3530 Prozessor. Die Kommunikationszeiten der Prozessoren untereinander werden direkt am SPI-, bzw. I2C-Port mit einem Oszilloskop gemessen. Die Ausführungszeiten der Bufferverarbeitung werden am Oszilloskop durch Ein- und Ausschalten von GPIO-Pins messbar.

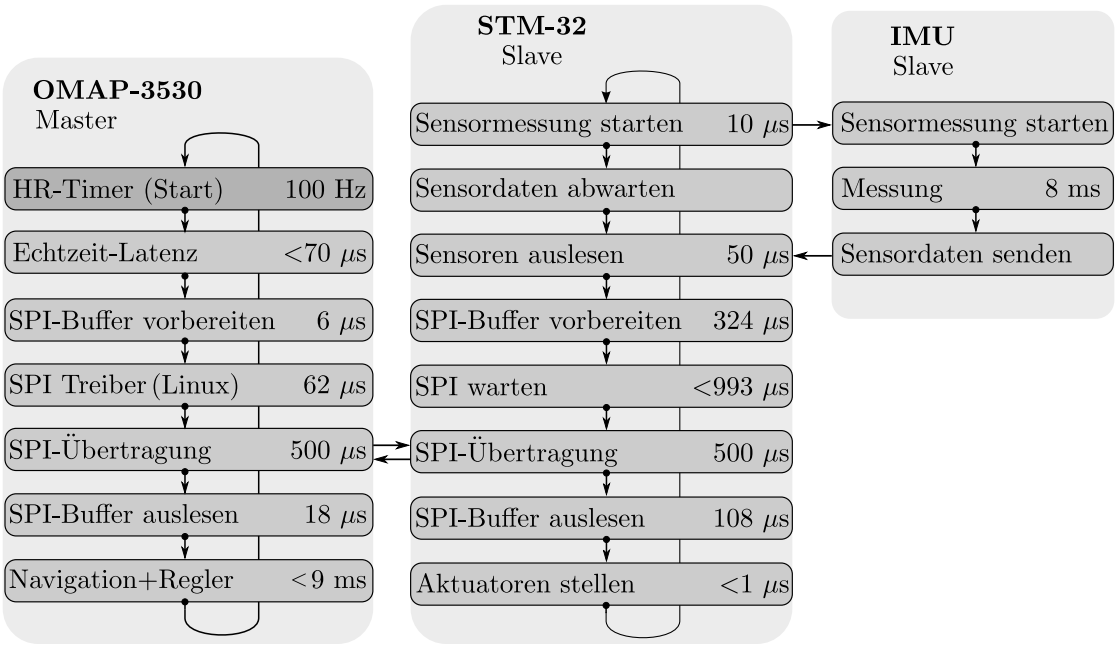


Abbildung 2.10.: Kommunikationsstruktur der Prozessoren und der IMU, 100 Hz Reglertakt.

Nach der SPI-Kommunikation verarbeitet der STM32-Prozessor die Signale und leitet diese weiter, z.B. an die Aktuatoren. Während der OMAP-3530 Prozessor bereits den nächsten Regleralgorithmus berechnet, initialisiert der „kleine“ Prozes-

[1]Fachlich korrekt handelt es sich beim „Reglerprozess“ um einen Posix-Thread.

sor die Sensormessung. Nach erfolgter Sensormessung wird der SPI-Buffer vorbereitet und der STM-32 Prozessor wartet auf den nächsten Reglertakt, also eine SPI-Übertragung des OMAP-3530.

Die Interrupt-Routinen für die Funkfernbedienungssignale und die Aktuatorsignalgenerierung des STM-32 Prozessors sind in Abbildung 2.10 nicht dargestellt. Diese Interrupts können vernachlässigt werden, da sie sich deutlich unterhalb des μs Bereichs befinden.

Beachtet man, dass ein Reglertakt des OMAP-3530 Prozessors 10 ms dauert und berücksichtigt man alle Verzögerungen, so kann man zusammenfassend sagen, dass vom Auslesen der Sensoren bis zum Stellen der Aktuatorsignale eine maximale Latenz von Δt_{max}=12,046 ms zu erwarten ist. Diese Latenz soll als zusätzliche Aktuatorverzögerung in den nachfolgenden Betrachtungen Beachtung finden.

Während der OMAP-3530 Prozessor mit dem Echtzeit-Linux als Betriebssystem arbeitet, wird beim STM-32 Prozessor auf ein Betriebssystem verzichtet. Dieser arbeitet als reiner Zustandsautomat und übergibt, im Fall eines Ausfalls des „großen“ Prozessors, die Aufgabe der Flugzeugsteuerung an den Piloten. D.h. die Funkfernbedienungssignale werden direkt an die Aktuatoren weitergeleitet. Diese Eigenschaft ist ein wichtiger Bestandteil zur Erfüllung der Modellflugrichtlinien, in denen der Pilot im Notfall jederzeit das UAV kontrollieren können muss.

Die Kommunikationsstruktur bietet noch viel Verbesserungspotenzial zur Verkürzung von Latenzen. Ein Verzicht auf den „kleinen“ STM-32 Prozessor kann zu einer Einsparung von Übertragungszeiten führen. Eine Berücksichtigung der tatsächlich nötigen Berechnungszeit eines Reglertaktes kann die Latenzen verkürzen. So ist es denkbar, die Aktuatorsignale bereits nach erfolgter Reglertaktberechnung zu stellen. Eine Erhöhung des Reglertaktes ist ebenfalls denkbar.

2.9 Implementierungskette

Die Geschichte der Regelungstechnik ist eng verknüpft mit dem Fortschritt der Implementierungskette. Bereits vor dem digitalen Zeitalter wurden in Flugzeugen elektromechanische Elemente zu Navigations- und Regelungszwecken genutzt. Als Beispiel sei die Rückführung der Lage eines mechanischen Kreisels im Rahmen einer Lageregelung genannt. Schon damals haben sich Schaltpläne als ein anschauliches Entwurfswerkzeug einer regelungstechnischen Anwendung etabliert. Der Schaltplan

ist aufgebaut aus verschiedenen Blöcken, z.B. einem Integrator oder einem Aktuator, die über Signale miteinander verknüpft sind. Dem Plan folgend, entsteht die praktische Umsetzung des Lagereglers bei der pre-digitalen Maschine durch mechanische Verknüpfung der Elemente, z.B. durch Verlöten von Kabeln zwischen den Elementen. Man kann sich leicht vorstellen, dass eine Designänderung im Schaltplan einen nicht unerheblichen Aufwand in der Neuanordnung der Element-Verknüpfung nach sich zieht.

Die Verfügbarkeit von Digitaltechnik hat den Implementierungsprozess drastisch verändert. Mit Mikroprozessoren waren erstmals modulare Elemente verfügbar, die die Umsetzung von Designänderungen ohne mechanische Neuverknüpfung erlaubten. So ist es allein durch Umprogrammieren des Prozessors möglich, z.B. einen Integrator zum System hinzuzufügen. Während der Implementierungsaufwand vor der Digitalisierung in der mechanischen Verknüpfung von Elementen lag, verschiebt sich der Implementierungsaufwand nun in die Entwicklung von Maschinencode. Die gewonnene Modularität erlaubt von da an die Entwicklung und Erprobung von Algorithmen, wie z.B. dem Kalman Filter. Allerdings bestimmt von diesem Zeitpunkt an die Programmiersprache den Faktor des Implementierungsaufwands. Der Schaltplan muss in den Maschinencode übersetzt werden.

In den letzten Jahrzehnten hat die Entwicklung von Personal-Computern die computergestützte Schaltplanentwicklung zum Quasi-Standard etabliert. Die Schaltpläne werden dabei in Simulationsumgebungen entworfen und können noch vor der Implementierung in einer Simulationsumgebung auf deren Funktionalität überprüft werden. Seit kurzer Zeit erlauben sogenannte „Autocode-Generatoren“ die direkte Übersetzung des Schaltplans in die Maschinensprache. Im Rahmen dieser Arbeit erfolgte die Verknüpfung eines solchen Autocode-Generators mit dem Autopiloten. Ab diesem Zeitpunkt ist die Programmierung einer flugfähigen Autopilotensoftware mit nur einem „Maus-Klick“ direkt aus dem Schaltplan möglich.

Letztendlich wird durch die Geschichte deutlich, dass mit der Automatisierung der Implementierungkette die Zeitressourcen des Entwicklers vollständig in den Schaltplanentwurf verlagert werden. Um die Unterschiede des Implementierungsaufwands zu verdeutlichen, werden im Folgenden die verschiedenen Programmiersprachen vorgestellt. Abschließend folgt die Darstellung der im Rahmen dieser Arbeit entstandenen Implementierungkette.

2.9.1 Programmiersprachen

Das folgende Unterkapitel stellt verschiedene Implementierungstechniken am Beispiel der Programmiersprachen Assembler, C, Matlab und Simulink dar. Die Hierarchie dieser Sprachen kann, wie im Bild 2.11 und den folgenden Ausführungen dargestellt, als Abstraktionsstufen der Maschinensprache interpretiert werden.

Abbildung 2.11.: Vergleich verschiedener Programmiersprachen.

Mit der Verfügbarkeit der ersten Mikroprozessoren entstand der Assembler als eine der ersten Programmiersprachen um 1952. Diese arbeitet direkt auf der Registerebene der Rechenmaschine. Im Gegensatz zur Maschinensprache erlaubt Assembler eine Übersicht über jede Rechenoperation, indem die Registeradressen (Zahlen) durch Namen substituiert werden und über die Syntax den Programmablauf darstellen. Als Weiterentwicklung des Assemblers etablierte sich die Programmiersprache „C“ im Jahr 1978 als Standardprogrammiersprache der Unix-Systeme. Am folgenden Beispiel wird dargestellt, wie der Programmieraufwand in der C-Sprache durch die Zusammenfassung mehrerer Assembler-Registeroperationen zu einem Satz reduziert wird. Eine Addition auf einem 8-Bit Prozessor läuft im Assembler folgendermaßen ab:

Assembler-Code:

```
ldi   r16, 50     :Lade Register 16 mit der Zahl 50
ldi   r17, 100    :Lade Register 17 mit der Zahl 100
add   r16, r17    :Addiere Register r16 und r17 und schreibe
                  :das Ergebnis in Register r16
```

Einen äquivalenten C-Code stellt das folgende Programm dar:

C-Code:
$x = 50 + 100;$

Dieser C-Code ist relativ zum Assembler intuitiv an die Sprache der üblichen Schul-Mathematik angelehnt und damit überschaubarer. Zwar kann Assembler die effizientere Programmiersprache im Bezug zur Rechenzeit sein, gleichzeitig ist der Zeitaufwand beim Programmieren in C geringer [Smi97]. Die Effizienz des C-Code ist stark abhängig von den Fähigkeiten des Programmierers, der Anpassung des C-Code an die Möglichkeiten des Prozessors sowie von den Optimierungseigenschaften des C-Compilers selbst. Im Fall weiterer Rechenoperationen, z.B. Vektormultiplikation gerät der C-Code allerdings an die Grenzen der Übersichtlichkeit. So wird die Vektormultiplikation $\vec{y} = \vec{x} * 5$ mit einer Schleife berechnet:

C-Code:

$for(i = 0; i < 10; i++)\{$	**:Zähle i von 0 bis 9**
$y[i] = x[i] \cdot 5;$	**:Multipliziere $\vec{x}$ mit 5 und speichere in $\vec{y}$**
$\}$	**:für alle 10 Elemente.**

Um diesem Problem entgegenzuwirken, wurde im Jahr 1984 Matlab als kommerzielle Software angeboten. Besonders im Bereich der Regelungstechnik ist Matlab eine der am meisten eingesetzten Mathematik-Sprachen. Im Gegensatz zum C-Code kann die Vektormultiplikation in Matlab wesentlich einfacher dargestellt werden.

Matlab-Code:

$y = x \cdot 5;$	**:Entspricht $\vec{y} = \vec{x} * 5$**

Parallel mit Matlab entstand Simulink (anfangs noch Simulab) als ein piktogrammbasiertes Programm zum computergestützten Entwurf von Schaltplänen. Mit dem folgenden Beispiel soll der simulierbare Schaltplanentwurf dargestellt werden. Das Simulink Diagramm in Abbildung 2.12 addiert eine Sinus Funktion zu einer Sprungfunktion. Diese Summe wird in ein Verzögerungsglied erster Ordnung geleitet und das Ergebnis mit einem Vektor „$Constant = [1, 2, 3]$“ addiert. Das Ergebnis ist in einem Diagramm, dem sogenannten Scope sichtbar. Diese Schaltpläne werden, je nach Betrachtungsweise, auch als Modelle bezeichnet.

Simulink:

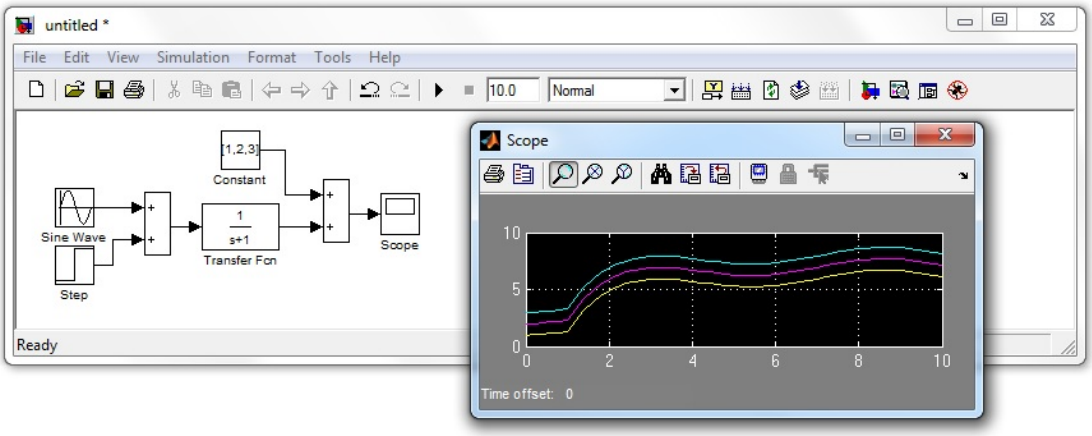


Abbildung 2.12.: Simulink Simulationsumgebung mit einem einfachen Schaltplan.

Im Gegensatz zum textbasierten Programm werden Signale als Verbindungen zwischen Blöcken (z.B. Übertragungsfunktion) dargestellt. Aus einer graphischen Bibliothek können Funktionen per Drag and Drop zwischen die Signale geschaltet werden. Der Anwender braucht keine Sprachkenntnisse in einer Programmiersprache.

2.9.2 Realtime Workshop Embedded Coder

Mit dem dargestellten Design-Werkzeug Simulink können Schaltpläne zwar simuliert werden, die Implementierung des Schaltplans z.B. in einem Autopiloten erfolgt bis jetzt aber noch manuell, also durch einen Programmierer. Mit dem sogenannten „Embedded-Coder“ ändert sich die Situation. Der Embedded Coder erlaubt indirekt das Erstellen eines flugbereiten Schaltplan-Programms, durch Bereitstellen einer entsprechenden C-Datei. In einem Simulink-Schaltplan existiert dazu für jeden Block eine Target-Language-Compiler (TLC) Datei, die die Übersetzung der Blöcke in eine C-Funktion definiert. Die im Schaltplan definierten Signal-Ein- und Ausgänge bilden die Ein- und Ausgänge der so entstandenen C-Funktion. Entsprechend dem Schaltplan werden dann die einzelnen C-Funktionen verknüpft und bilden so eine Regler.c Datei. Letztendlich kann mit einem C-Compiler die Übersetzung in Maschinensprache erfolgen, wie in Abbildung 2.13 stark vereinfacht dargestellt.

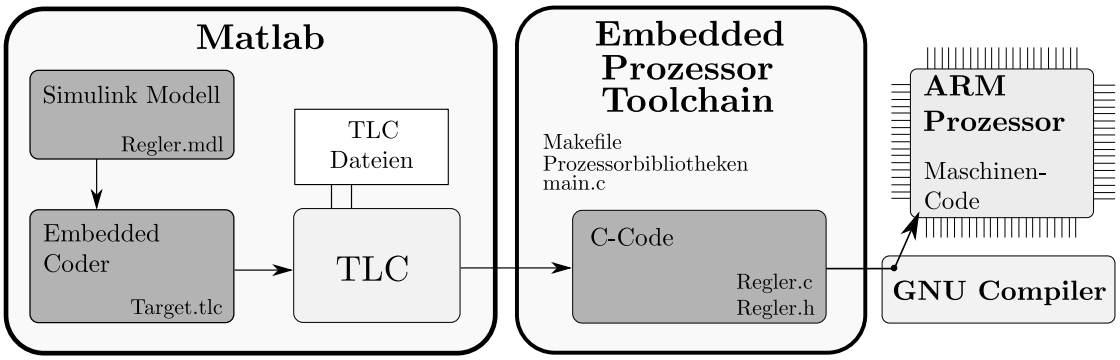


Abbildung 2.13.: Programmiervorgang vom Simulink Schaltplan (bzw. Modell) zum Maschinencode.

Für einen flugfertigen Autopiloten ist noch die Anbindung der Aktuatorik, bzw. Sensorik notwendig. Diese Anbindung erfolgt wieder über Blöcke, z.B. einen „Servo-Block“. Die dem Block entsprechende TLC-Datei muss allerdings einmal manuell für die entsprechende Autopilotenhardware, z.B. den Servo, erstellt werden.

Da die Implementierung der einzelnen Funktionen in C-Code direkt aus einer dem Simulink Block zugeordneten Datei erfolgt, kann keine globale Aussage über die Effizienz des generierten C-Codes gemacht werden. Die Entkopplung des Programmierers vom Schaltplan-Designer hat allerdings einen Vorteil: Die C-Code-Güte im Bezug zur Code-Effizienz ist unabhängig von den Fähigkeiten des Designers. Für einfache Reglerstrukturen wird z.B. eine Bibliothek mit Integratoren, Verzögerungsgliedern, Lookup-Tabellen usw. verwendet. Da aber für jeden Block eine individuelle TLC Datei existiert, kann der generierte C-Code beliebig optimiert werden.

Um einen Eindruck des Zusammenhangs zwischen C-Code und Schaltplan zu vermitteln, wird beispielhaft am Simulink Schaltplan in Abbildung 2.14, der vom Embedded-Coder resultierende C-Code dargestellt.

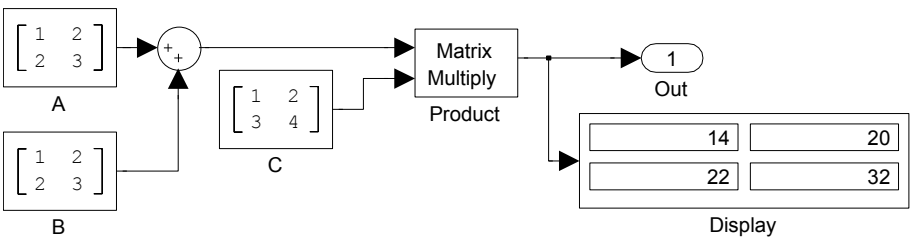


Abbildung 2.14.: Embedded Coder Beispiel.

Resultierender C-Code

```
tmp[0] = A[0] + B[0];
tmp[1] = A[1] + B[1];
tmp[2] = A[2] + B[2];
tmp[3] = A[3] + B[3];
for (i = 0; i < 2; i++) {
  Out[i] = 0.0;
  Out[i] = tmp[i] * C[0] + Out[i];
  Out[i] = tmp[i + 2] * C[1] + Out[i];
  Out[i + 2] = 0.0;
  Out[i + 2] = Out[i + 2] + tmp[i] * C[2];
  Out[i + 2] = tmp[i + 2] * C[3] + Out[i + 2];
}
```

Zusammenfassend kann man sagen, dass die Sprache der Entwicklungsumgebung einen direkten Einfluss auf die Anforderungen des Programmierers und auf den Programmieraufwand hat. Gleichzeitig ist erkennbar, dass mit jeder „Stufe“ des Beispiels von Bild 2.11 ein fließender Übergang vom Programmierer zum Designer stattfindet. Die Automatisierung des Programmiervorgangs direkt aus dem Design des Schaltplans hat den wesentlichen Vorteil, dass der Designer seinen Entwurf direkt aus der Entwicklungsumgebung mit nur einem „Maus-Klick“ auf der Zielhardware testen kann. Die Entwicklung in Simulink gegenüber C-Code ist eine Weiterentwicklung vergleichbar mit dem Schritt von Assembler zu C.

2.9.3 Integrationsverfahren

Das grundlegende Ziel einer Flugdynamikmodellierung ist es, die Flugbewegungen so realistisch wie möglich darzustellen. Um der Bandbreite der Anwendungen gerecht zu werden, existieren verschiedene Implementierungen von Flugmodellen. All diesen Modellen gemein ist die Anwendung von Newtons zweitem Gesetz auf ein Differentialgleichungssystem mit sechs Freiheitsgraden. Bekanntermaßen kann die Wahl des Integrationsverfahrens sowohl einen Einfluss auf die Genauigkeit der Lösung von Differentialgleichungen, als auch auf die benötigten Rechenressourcen haben. Im zeitdiskreten Fall stehen mit dem Embedded Coder mehrere Integrationsverfahren zur Verfügung: Euler (ode1), Heun (ode2), Bogacki-Shampine (ode3) [BS89], Runge-Kutta (ode4) und Dormand-Prince (ode5 und ode8) [DP77]. Zur weiteren Vertiefung

der Thematik der Integrationsverfahren wird an dieser Stelle auf [HNW93] verwiesen. Im Laufe dieser Arbeit wird der Bezug zwischen Integrationsverfahren und Prozessorlast weiter untersucht.

2.9.4 Diskussion

Zusammenfassend kann man sagen, dass hier eine Hardware-Softwarestruktur geschaffen wurde, um Latenzen möglichst berechenbar und kurz zu halten. Der Implementierungsaufwand kann mit Autocode-Verfahren deutlich reduziert werden. Allerdings können die Latenzen im Bereich Betriebssystem und Prozessorkommunikation noch weiter reduziert werden. Gleichzeitig ist die Gesamtsystembetrachtung von UAV-Autopilotenstrukturen, als wissenschaftliche Disziplin, kaum etabliert. Der oben dargestellte Entwurf verdeutlicht, dass eben die Erforschung und Entwicklung dieser Systemstrukturen ein signifikantes Verbesserungspotenzial im Bezug auf die Eigenschaften von Autopilotenkomponenten, wie z.B. Navigation und Regelung, haben kann. Zudem sind Erkenntnisse aus diesen Bereichen nicht nur im Bereich der Flugzeuge, sondern im Großteil des Themengebietes Robotik anwendbar.

Wenn man bedenkt, dass die hier entwickelten Autopilotenhardware nicht nur auf Modellflugzeugen, sondern auch auf Modellautos, Modellbooten, Quadrocoptern und anderen „Modellbau-Spielzeugen“ angewendet werden kann, wird das Potenzial der piktogrammbasierten Schaltplanentwicklung besonders deutlich. Nur durch Strukturänderungen im Schaltplan können die, im Rahmen dieser Arbeit entstandenen Reglerstrukturen auch auf die anderen Anwendungen übertragen werden. All diese „Spielzeuge“ sind meist mit Standard-Servos ausgestattet und können ohne weitere Änderungen mit dem hier entstandenen Autopiloten gesteuert werden.

3 Simulation, Navigation und Bahnregler

Vor der Durchführung von praktischen Versuchen haben sich Simulationen als probates Werkzeug zur Erforschung von Systemverhalten durchgesetzt. Tatsächlich findet die gesamte Entwicklung eines Reglers oder Navigationsblocks üblicherweise in einer Simulation statt, bevor dieser im realen Versuch untersucht wird. Im Zentrum der Flugsimulation steht die Flugzeugdynamik, die mit sechs Freiheitsgraden die Bewegung des Flugzeugs im Raum definiert. Die Simulation geht allerdings weit über die Flugzeugdynamik hinaus und berücksichtigt weitere Elemente, die einen geschlossenen Kreis bilden. Dabei werden z.B. auch das Wind- und Turbulenzmodell, die Navigationsalgorithmen, die Reglerstrukturen und die Verzögerungen und Begrenzungen der Aktuatoren und Sensoren beachtet. Einleitend wird die Verknüpfung der einzelnen Elemente in Abbildung 3.1 als geschlossener Regelkreis dargestellt.

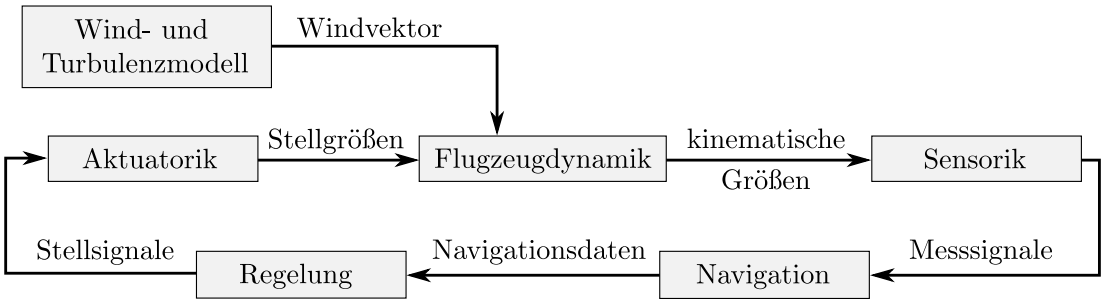


Abbildung 3.1.: Darstellung der Flugzeugsimulation als geschlossener Regelkreis.

Die Definition der Elemente bedarf einer genaueren Betrachtung, da diese die Basis der späteren Versuche und der zu untersuchenden Reglerarchitekturen bildet. Während im letzten Kapitel bereits die Elemente Aktuatorik und Sensorik erwähnt wurden, stehen in diesem Kapitel die verbleibenden Bestandteile der Simulation im Fokus. Das Element „Regelung“ soll in seinen Variationen im späteren Verlauf der Arbeit weiter untersucht werden.

3.1 Windmodell

Die auf das Flugzeug wirkenden Kräfte und Momente sind eine Funktion der relativen Bewegung des Flugzeugs zu dem ihm umgebenden Medium, der Luft. In einer

ersten Annahme kann die Luft als ruhende Masse aufgefasst werden, womit die Anströmung des Flugzeugs lediglich aus dem Geschwindigkeitsvektor des Flugzeugs selbst resultiert. Dies entspricht einem störungsfreien Flug, bildet aber nicht immer die Realität ab. Zur Generierung adäquater Störgrößen wird die Luft gegenüber der Erde im sogenannten Windmodell bewegt. Da die Bewegungen der Luftmassen allerdings in dessen Komplexität nicht nachzubilden sind, wird das atmosphärische Strömungsfeld nach [BAL11] unter bestimmten Annahmen vereinfacht. Dabei wird der Wind in zwei Anteile gespalten: Einen „Wind“ der seine Richtung und Stärke gar nicht, oder nur langsam ändert und einer „Turbulenz“, die als hochfrequentes Chaos bezeichnet werden kann.

Eine Modellierung der Turbulenz als stochastischer Prozess bietet sich mit dem Dryden Spektrum an. Dazu wird das weiße Rauschen R mit einer Übertragungsfunktion (Zeitkonstante T_i) moduliert:

$$\vec{V}_r = \begin{bmatrix} u_r \\ v_r \\ w_r \end{bmatrix} = \begin{bmatrix} \sqrt{2 \cdot \sigma_u^2 \cdot T_u} \cdot (T_u \cdot s + 1)^{(-1)} \cdot R \\ \sqrt{\sigma_v^2 \cdot T_v} \cdot (1 + s \cdot \sqrt{3} \cdot T_v) \cdot (T_v \cdot s + 1)^{(-2)} \cdot R \\ \sqrt{\sigma_w^2 \cdot T_w} \cdot (1 + s \cdot \sqrt{3} \cdot T_w) \cdot (T_w \cdot s + 1)^{(-2)} \cdot R \end{bmatrix}. \tag{3.1}$$

Durch Addition der Turbulenzen mit den konstanten Windanteilen $\vec{V}_c = [u_c, v_c, w_c]^T$ ergibt sich für den Windvektor im geodätischen Koordinatensystem (Indize g):

$$\vec{V}_{W_g} = \begin{bmatrix} u_r + u_c \\ v_r + v_c \\ w_r + w_c \end{bmatrix}. \tag{3.2}$$

Mit der Rotationsmatrix $\boldsymbol{M}_{fg}(\Phi, \Theta, \Psi)$ wird der lokale Windvektor in das flugzeugfeste Koordinatensystem (Indize f) transformiert. Diese Rotationsmatrix ist mit der Drehfolge der Eulerwinkel $\Psi \rightarrow \Theta \rightarrow \Phi$ nach LN9300 definiert zu:

$$\boldsymbol{M}_{fg} = \boldsymbol{M}_{gf}^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\Phi & sin\Phi \\ 0 & -sin\Phi & cos\Phi \end{bmatrix} \begin{bmatrix} cos\Theta & 0 & -sin\Theta \\ 0 & 1 & 0 \\ sin\Theta & 0 & cos\Theta \end{bmatrix} \begin{bmatrix} cos\Psi & sin\Psi & 0 \\ -sin\Psi & cos\Psi & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.3}$$

womit sich der lokale Windvektor im Koordinatensystem des Flugzeugs nach folgender Gleichung berechnet:

$$\vec{V}_{W_f} = \boldsymbol{M}_{fg} \cdot \vec{V}_{W_g}. \tag{3.4}$$

3.2 Nichtlineare Flugzeugdynamik

Die Flugzeugbewegung im Raum wird üblicherweise als System mit sechs Freiheitsgraden dargestellt [BAL11, Hol04, Krü12]. Dieses System wird dabei in eine translatorische und die rotatorische Gruppe unterteilt. Weiterhin wird mit zwölf nichtlinearen Differentialgleichungen der Zusammenhang zwischen Kräften, Geschwindigkeiten und Positionen, bzw. Momenten, Drehraten und Lagewinkeln gebildet. Diese Differentialgleichungen beschreiben also die zeitliche Änderung der Zustände des Fluggerätes. Die folgenden Zusammenhänge werden angelehnt an [BAL11] beschrieben.

Stellt man sich z.B. vor, dass kleinste Luft-Turbulenzen hochkomplexe Schwingungsmuster am Flügel erzeugen können, so wird deutlich, dass das System der Flugzeugbewegung nur unter vereinfachenden Annahmen nachgebildet werden kann.

Vereinfachende Annahmen

- Das Flugzeug wird als Starrkörper mit konstanter Masse angesehen. Schwingungen und Bewegungen der Flugzeugkomponenten untereinander werden so nicht berücksichtigt.
- Erdrotation und Flugzeugbewegung sind so klein, dass aus der Erdrotation und Erdkrümmung entstehenden Zentripetal- und Corioliskräfte vernachlässigt werden können. Die Erde kann in diesem Fall als eine flache, ruhende „Scheibe“ betrachtet werden. Auch wenn in diesem Abschnitt der Indize (g) des geodätischen Koordinatensystems Anwendung findet, gelten diese Annahmen nicht für den Abschnitt der Navigationsalgorithmen.
- Das Flugzeug ist in dessen x-z-Ebene symmetrisch.
- Die Anströmung wird als quasi-stationär angenommen.
- Die auf das Flugzeug wirkenden Kräfte greifen im Flugzeugschwerpunkt an.

Zur Beschreibung der translatorischen Flugzeugbewegung wird mit dem Impulserhaltungssatz das Kräftegleichgewicht im Flugzeugschwerpunkt sowie im Koordinatensystem des Flugzeugs gebildet. Die Summe der am Flugzeug angreifenden Kräfte, die Triebwerkskraft $\vec{F}_{TW,f} = [X^F, Y^F, Z^F]^T$, die aerodynamischen Kräfte $\vec{R}_e^A = [X^A, Y^A, Z^A]^T$ und die Gewichtskraft $\vec{G}_f = \boldsymbol{M}_{fg} \cdot [0, 0, m \cdot g]^T$ werden dazu dem Impuls gleichgesetzt:

$$m \cdot \left(\frac{d\vec{V}_K}{dt} \right)_f^g = \vec{F}_{TW,f} + \boldsymbol{M}_{fe} \vec{R}_e^A + \vec{G}_f. \tag{3.5}$$

Dabei wird berücksichtigt, dass in Folge von Windkanalmessungen, die aerodynamischen Kräfte meist im experimentellen Koordinatensystem angegeben werden. Daraus folgt, dass eine Transformation dieser Kräfte in das flugzeugfeste Koordinatensystem mit der Rotationsmatrix $\boldsymbol{M}_{fe}$ nach LN9300 nötig ist. Unter Berücksichtigung, dass die Bahngeschwindigkeit $\vec{V}_K$ gegenüber dem erdfesten Koordinatensystem abgeleitet wird, ergibt sich für die translatorische Geschwindigkeits-Differentialgleichung:

$$\left(\frac{d\vec{V}_K}{dt} \right)_f^f = \begin{bmatrix} \dot{u}_K \\ \dot{v}_K \\ \dot{w}_K \end{bmatrix}_f = \frac{1}{m} \boldsymbol{M}_{fe} \begin{bmatrix} X^A \\ Y^A \\ Z^A \end{bmatrix}_e + \frac{1}{m} \begin{bmatrix} X^F \\ Y^F \\ Z^F \end{bmatrix}_f + \begin{bmatrix} -sin\Theta \\ sin\Phi cos\Theta \\ cos\Phi cos\Theta \end{bmatrix} g - \begin{bmatrix} p_K \\ q_K \\ r_K \end{bmatrix}_f \times \begin{bmatrix} u_K \\ v_K \\ w_K \end{bmatrix}_f . \tag{3.6}$$

Durch Integration der Bahngeschwindigkeit wird letztendlich die Position nach Gleichung (3.7) errechnet. Üblicherweise findet die Position erst im erdfesten Koordinatensystem eine sinnvolle Verwendung, z.B. in Verbindung mit GPS-Daten. Daher wird die flugzeugfeste Bahngeschwindigkeit zuvor mit der Rotationsmatrix $\boldsymbol{M}_{gf}$ transformiert.

$$\frac{d\vec{s_g}}{dt} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}_g = \begin{bmatrix} u_K \\ v_K \\ w_K \end{bmatrix}_g = \boldsymbol{M}_{gf} \cdot \begin{bmatrix} u_K \\ v_K \\ w_K \end{bmatrix}_f . \tag{3.7}$$

Damit sind die ersten sechs Differentialgleichungen der Translationsbewegung festgelegt. Die Herleitung der Rotationsbewegung findet in ähnlicher Weise mit dem Drehimpulssatz aus den im Schwerpunkt angreifenden Momenten $\vec{M}_f$ statt:

$$\boldsymbol{T}_f \cdot \left(\frac{d\vec{\omega}_f^{gf}}{dt} \right)_f^g = \sum \vec{M}_f. \tag{3.8}$$

Darin enthalten ist der Trägheitstensor des symmetrischen Flugzeugs:

$$\boldsymbol{T}_f = \begin{bmatrix} I_{xx} & 0 & -I_{zx} \\ 0 & I_{yy} & 0 \\ -I_{zx} & 0 & -I_{zz} \end{bmatrix}. \tag{3.9}$$

Beachtet man wieder, dass die Ableitung bezüglich des erdfesten Koordinatensystems formuliert ist, erweitert sich die Gleichung um den sogenannten Euler-Term $\vec{\omega}_f^{gf} \times (\boldsymbol{T}_F \cdot \vec{\omega}_f^{gf})$. Umgestellt nach der Ableitung der Drehgeschwindigkeit ergibt sich der Zusammenhang:

$$\dot{\vec{\omega}}_f^{gf} = \boldsymbol{T}_f^{-1} \cdot \left[\sum \vec{M}_f - \vec{\omega}_f^{gf} \times (\boldsymbol{T}_F \cdot \vec{\omega}_f^{gf}) \right]. \tag{3.10}$$

Ähnlich wie bei der Translation ist auch die Summe der Momente aus einem Momentenvektor des Triebwerks $\vec{Q}_f^F = [L^F, M^F, N^F]_f^T$ und einem aerodynamischen Momentenvektor $\vec{Q}_f^A = [L^A, M^A, N^A]_f^T$ aufgebaut:

$$\sum \vec{M}_f = \vec{Q}_f^F + Q_f^A. \tag{3.11}$$

Formuliert man Gleichung 3.10 aus, so resultiert für die Ableitung der Drehgeschwindigkeit:

$$\dot{\vec{\omega}}_f^{gf} = \begin{bmatrix} \dot{p}_K \\ \dot{q}_K \\ \dot{r}_K \end{bmatrix}_f = \mathbf{T}_f^{-1} \left\{ \begin{bmatrix} L^A + L^F \\ M^A + M^F \\ N^A + N^F \end{bmatrix}_f - \begin{bmatrix} q_K r_K (T_z - T_y) - p_K q_K T_{xz} \\ r_K p_K (T_x - T_z) + (p_K^2 + r_K^2) T_{xz} \\ p_K q_K (T_y - T_x) + q_K r_K T_{xz} \end{bmatrix}_f \right\}. \tag{3.12}$$

Bei der Ableitung der Lage wird mit der Transformationsmatrix $\boldsymbol{M}_{\Phi f}$ berücksichtigt, dass die Lage im erdfesten Koordinatensystem definiert ist, während die Drehraten mit dem Koordinatensystem des Flugzeuges verknüpft sind. So kann nach Gleichung 3.13 die Lage mit dem Rollwinkel Φ, dem Nickwinkel Θ und dem Gierwinkel Ψ berechnet werden.

$$\left(\frac{d\vec{\Omega}}{dt} \right) = \begin{bmatrix} \dot{\Phi} \\ \dot{\Theta} \\ \dot{\Psi} \end{bmatrix} = \boldsymbol{M}_{\Phi f} \vec{\omega}_{KF} = \begin{bmatrix} 1 & sin\Phi tan\Theta & cos\Phi tan\Theta \\ 0 & cos\Phi & -sin\Phi \\ 0 & sin\Phi / cos\Theta & cos\Phi / cos\Theta \end{bmatrix} \cdot \begin{bmatrix} p_K \\ q_K \\ r_K \end{bmatrix}_f. \tag{3.13}$$

Damit sind die zwölf Differentialgleichungen für Translation und Rotation festgelegt. Durch numerische Integration können aus diesen Gleichungen die Position, Geschwindigkeit, Lage und die Drehraten aus Kräften und Momenten bestimmt werden. Im Rahmen einer Simulation berechnen sich die Kräfte und Momente, neben steuerbaren Größen, wieder aus der Flugzeuggeschwindigkeit und der Lage, so dass ein geschlossener Simulationskreislauf entsteht. Bei geeigneter Parameterwahl kann man sagen, dass das Flugzeug bereits in der Simulation fliegen kann.

Während die aerodynamischen Kräfte mit den folgenden Untersuchungen keinen tieferen Zusammenhang bilden, wird an dieser Stelle weiter an [BAL11] verwiesen. Da im späteren Verlauf dieser Arbeit die nichtlineare Regelung der Lage- und Rotationsdynamik erfolgt, bedarf die Momentenzuweisung allerdings einer weiteren Betrachtung.

3.2.1 Momentenzuweisung

Die Momentenzuweisung stellt den direkten Zusammenhang zwischen den Momenten und der Anströmung des Flugzeugs her, also dem Anstellwinkel α, dem Schiebewinkel β und dem Staudruck $\bar{q}$. Letzterer berechnet sich nach der Formel

$$\bar{q} = \rho/2 \cdot V_A^2, \tag{3.14}$$

mit der Fluggeschwindigkeit V_A und der Umgebungsluftdichte ρ. Da die Fluggeschwindigkeit gegenüber der dem Flugzeug umgebenden Luft definiert ist, bildet sich nach Gleichung 3.15 und 3.16 der Zusammenhang zum bereits dargestellten Windmodell:

$$V_A = \sqrt{(u_{Af}^2 + v_{Af}^2 + w_{Af}^2)}, \tag{3.15}$$

wobei $\vec{V}_A = [u_A, v_A, w_A]_f^T$ dem Fluggeschwindigkeitsvektor in körperfesten Koordinaten entspricht. Unter Berücksichtigung des Windvektors ergibt sich der Fluggeschwindigkeitsvektor als Differenz zum Bahngeschwindigkeitsvektor:

$$\vec{V}_A = \vec{V}_K - \vec{V}_W. \tag{3.16}$$

Der Anstellwinkel α und der Schiebewinkel β berechnen sich entsprechend Gleichung 3.17 und 3.18 aus den Anteilen des Fluggeschwindigkeitsvektors:

$$\alpha = \arctan \frac{w_{Af}}{u_{Af}}, \tag{3.17}$$

$$\beta = \arcsin \frac{v_{Af}}{V_A}. \tag{3.18}$$

Zur weiteren Vereinfachung der Modellbildung wird ein momentenfreier Antrieb angenommen:

$$\begin{bmatrix} L^A \\ M^A \\ N^A \end{bmatrix}_f = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}. \tag{3.19}$$

Mit dieser Vereinfachung hängen die im Flugzeug erzeugten Momente größtenteils vom Staudruck, der Flügelfläche S, der Bezugsflügeltiefe c, der Spannweite b und den Momentenbeiwerten C_i ab, wie in Gleichung 3.20 dargestellt.

$$\begin{bmatrix} L^F \\ M^F \\ N^F \end{bmatrix}_f = \bar{q} \cdot S \cdot \begin{bmatrix} C_L \cdot \frac{b}{2} \\ C_M \cdot c \\ C_N \cdot \frac{b}{2} \end{bmatrix}. \tag{3.20}$$

Nach [Rob13] wird die Konfiguration aus Gleichung 3.21 zur Berechnung der Momentenbeiwerte angewendet. Deutlich zu erkennen sind die Einflüsse des linken und rechten Querruders ξ_{li}, bzw. ξ_{re} und des Höhenruders η:

$$\begin{bmatrix} C_L \\ C_M \\ C_N \end{bmatrix}_f = \begin{bmatrix} C_{L,\beta} \cdot \beta + C_{L,p} \cdot p + C_{L,r} \cdot r + C_{L,\xi_{li}} \cdot \xi_{li} + C_{L,\xi_{re}} \cdot \xi_{re} \\ C_{M_0} + C_{M,\alpha} \cdot \alpha + C_{M,q} \cdot q + C_{M,\eta} \cdot \eta \\ C_{N,\beta} \cdot \beta + C_{N,p} \cdot p + C_{N,r} \cdot r + C_{N,\xi_{li}} \cdot \xi_{li} + C_{N,\xi_{re}} \cdot \xi_{re} \end{bmatrix}. \tag{3.21}$$

3.2.2 Bestimmung der Derivate

Basierend auf den Arbeiten in [Rob13] werden die Beiwertderivate des T200 den numerischen Berechnungen der Programme AVL (Athena Vortex Lattice) und Digital Datcom entnommen. Dabei werden die Derivate aus den geometrischen Abmessungen des T200 entweder mit einem potenzialtheoretischen Wirbelleitverfahren (AVL) oder einer Sammlung aus analytischen und semi-empirischen Methoden (Datcom) bestimmt. Da die Derivate eine starke Geschwindigkeitsabhängigkeit aufweisen, werden diese nicht als feste Werte, sondern als geschwindigkeitsabhängige Parameter definiert. In der Simulation wird dann, je nach Fluggeschwindigkeit, zwischen den einzelnen Derivaten interpoliert. Dies geschieht, wie in Abbildung 3.2 beispielhaft dargestellt, in Form von Lookup-Tabellen, die im Schaltplan als Blöcke dargestellt werden.

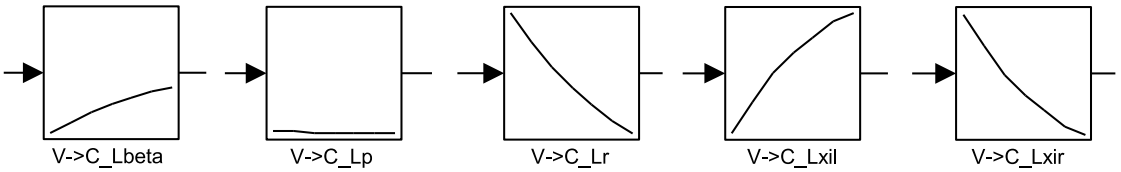


Abbildung 3.2.: Darstellung der Momentenbeiwerte-Lookup-Tabellen in Simulink.

3.3 Integriertes Navigationssystem

Das bisher beschriebene Flugzeug ist zwar bereits in der Lage zu fliegen, aber ohne Navigation und Regelung schwebt dieses bestenfalls sinnlos im virtuellen Raum. Eine Teillösung des Problems mit den vorhandenen Sensoren bietet das sogenannte integrierte Navigationssystem.

Das Grundprinzip der integrierten Navigationssysteme lässt sich nach [Win06] auf die alten Navigationstechniken der Seefahrt zurückführen. Schon damals wurde über das Aufintegrieren der Geschwindigkeit über den Kurs die Position bestimmt, wobei die Geschwindigkeitsmessung mit dem Log fehlerbehaftet ist. Die Messfehler führen zu einem Drift in der berechneten Position, d.h. der Positionsfehler nimmt mit der Zeit zu. Erst durch Absolutmessungen, also der Stützung der Positionsberechnung mit Landmarken, ist eine zuverlässige Navigation möglich. Jetzt stellt sich die Frage, wozu überhaupt die fehlerbehaftete, driftende Positionsberechnung benötigt wird? Diese ungefähre Positionsbestimmung erlaubt, trotz Fehler, die Überbrückung längerer Strecken und vereinfacht deutlich die Identifizierung der Landmarken auf der Karte, wenn der Navigator weiß, wo er sich ungefähr befindet. Gleichzeitig wird mit einer neuen Absolutposition der Drift der integrierten Position sichtbar, z.B. durch Annahme einer Strömung, die die Log-Messung stört. Das bedeutet, dass beide Messprinzipien sich ergänzende Eigenschaften besitzen.

Eines ähnlichen Prinzips bedienen sich die integrierten Navigationssysteme. Mit den Messdaten der IMU, also Drehraten und Beschleunigungen, wird eine Lage und eine Position aufintegriert. Da diese Sensoren über Messfehler verfügen, driftet die berechnete Lage und Position mit der Zeit von den tatsächlichen Werten ab. Mit der Stützung anhand von GPS Daten, also einer Absolutposition, erfolgt die Korrektur der integrierten Werte. Die Nachteile beider Messsysteme werden also durch komplementäre Eigenschaften ergänzt. Die GPS-Messung liefert eine zeitlich niedrig aufgelöste Position ohne Drift, hingegen ist mit der IMU eine hochdynamische Abbildung der Flugbahn möglich, diese besitzt aber keine Langzeitgenauigkeit.

Kalman Filter

Der Kalman Filter ist ein bewährtes Verfahren zur Datenfusion von GPS und IMU. Nach [Cra13, Win06] wird unterschieden zwischen „Tightly Coupled“ und „Loosely Coupled“, linear und nichtlinear. Daneben sind viele Variationen des Kalman Filters im Navigationsbereich untersucht worden, z.B. Self-Tuning Kalman Filter [Ban89],

Ensemble Kalman Filter [Gup09], Switching Kalman Filter [DMC00], um nur einige wenige zu nennen. Weiterhin können adaptive Elemente in Form von neuronalen Netzen mit dem Kalman Filter gekoppelt werden [CW03].

Im Rahmen dieser Arbeit kommt der Extended Kalman Filter (EKF) nach [Cra13] zum Einsatz. Dieser Filter hat sich in den hier erfolgten Fluguntersuchungen bewährt und bildet einen guten Kompromiss zwischen Ressourcenverbrauch und Genauigkeit. Im Wesentlichen kann dieser Filter in zwei Schritte unterteilt werden, die Propagation und das Update, wie in Abbildung 3.3 dargestellt. Mit jeder IMU-Messung werden die Drehraten $\vec{\omega}_f^{gf}$ und die Beschleunigungsdaten $\vec{a}_f$ im sogenannten Strapdown-Algorithmus zu einer Lage $\hat{\vec{q}}$, Geschwindigkeit $\hat{\vec{V}}_K$ und Position $\vec{s}_g$ im geodätischen System G (WGS84), bzw. körperfesten System K, aufintegriert. Der Hut-Indize symbolisiert die a-priori integrierten Zustände $\hat{\vec{x}}$ der Propagation. Daneben werden mit dem Kalman Filter weitere Zustände für Gyroskopbiase $\vec{b}_g$ und Beschleunigungssensorbiase $\vec{b}_a$ geschätzt, woraus sich die Zustände des Filters ergeben:

$$\vec{x} = \left[\vec{s}_g, \vec{q}, \vec{V}_K, \vec{b}_g, \vec{b}_a\right]^T . \tag{3.22}$$

Dabei wird die Lage in Quaternionen $\vec{q} = [q_0, q_1, q_2, q_3]$ innerhalb des Filters dargestellt. Abweichend zu [Cra13] wird, um CPU-Ressourcen einzusparen, auf die Schätzung der Skalierungsfaktoren verzichtet.

Das Update des Filters wird mit der GPS-Messung der Position $\vec{s}_{G,GPS}$ im geodätischen System (WGS84) gestartet und liefert Korrekturwerte $\Delta\vec{x}$ der Zustandswerte des Strapdown-Algorithmus, bzw. der Propagation. Zur detaillierten Beschreibung und Untersuchung des verwendeten Filters wird an dieser Stelle auf [Cra13] verwiesen.

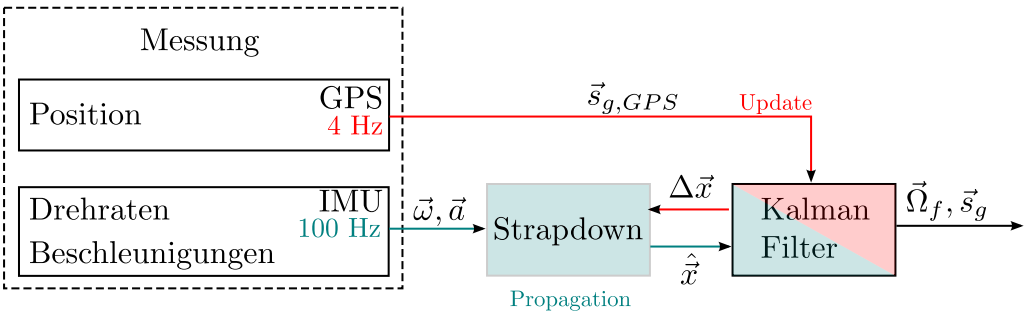


Abbildung 3.3.: Propagation und Update des Kalman Filters.

3.4 Bahnregelung - Definition der Sollgrößen

Mit der Navigation ist es bereits möglich, die Lage und die Position in zeitlich hoher Auflösung zu bestimmen. Ein Abfliegen einer gewünschten Bahn wird allerdings erst mit einem Bahnregler möglich. Es gibt viele Ansätze zur Definition von Flugbahnen. Klassische Ansätze der Luftfahrt basieren auf Abschnitten konstanter Bahnkrümmung, bzw. Geraden [Sch08]. Die Übergänge dieser klassischen Bahnen werden dabei meist nicht betrachtet. Gerade bei unbemannten Flugzeugen können die Bahnübergänge zu sprunghaften Änderungen im Hängewinkel führen, was sich z.B. bei Bildüberwachungsaufgaben negativ auswirken kann [Sch08]. Im UAV-Bereich bieten sich daher kubische Bezier-Spline-Kurven nach [Mac04] an, diese besitzen im Gegensatz zur klassischen Bahnregelung wesentliche Vorteile. So sind diese Spline-Kurven zweimal stetig differenzierbare Funktionen. Die damit verbundene Berechenbarkeit der Krümmung erlaubt eine Voraussage des zu fliegenden Hängewinkels. Weiterhin sind die Stützstellen, also die Punkte an denen zwei Spline-Kurven miteinander verbunden sind, mit der Definition einer Übergangsbedingung stetig ableitbar.

Anhand von Abbildung 3.4 wird das Prinzip der Spline-Kurven beschrieben. Der Start- und Endpunkt einer Spline N wird mit den Punkten $\vec{P}_0^N$ bzw. $\vec{P}_3^N$ dargestellt. Die Stützpunkte $\vec{P}_1^N$ und $\vec{P}_2^N$ definieren die Krümmung der Spline. Der Startpunkt $\vec{P}_0^{N+1}$ der darauffolgenden Spline $N+1$ wird mit dem Endpunkt $\vec{P}_3^N$ gleichgesetzt. Liegt zudem der Stützpunkt $\vec{P}_1^{N+1}$ auf einer Geraden mit dem Stützpunkt $\vec{P}_2^N$ der vorangehenden Spline, so ist die einmalige Ableitbarkeit an der Verbindungsstelle garantiert.

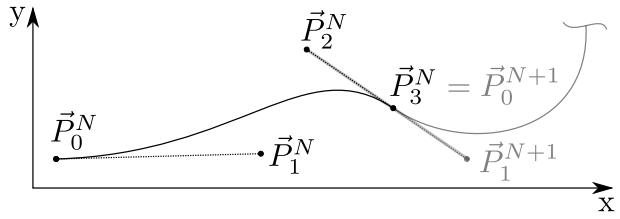


Abbildung 3.4.: Beispiel einer Bezier-Spline mit Kontrollpunkten zur Bahnführung.

Der Funktionsverlauf $\vec{P}(t) = [P_x(t), P_y(t)]^T$ einer Spline wird mit der Laufzeitvariablen $t = [0..1]$ in Gleichung 3.23 dargestellt. Das bedeutet, mit jedem Splineabschnitt wird die Laufzeitvariable t im Intervall $[0..1]$ wiederholt durchlaufen.

$$\vec{P}(t) = \vec{c}_3 \cdot t^3 + \vec{c}_2 \cdot t^2 + \vec{c}_1 \cdot t + \vec{c}_0 \quad \text{mit} \quad \begin{cases} \vec{c}_0 = \vec{P}_0 \\ \vec{c}_1 = -3\vec{P}_0 + 3\vec{P}_1 \\ \vec{c}_2 = 3\vec{P}_0 - 6\vec{P}_1 + 3\vec{P}_2 \\ \vec{c}_3 = -\vec{P}_0 + 3\vec{P}_1 - 3\vec{P}_2 + \vec{P}_3 \end{cases} \tag{3.23}$$

Zur Realisierung der Bahnführung wird der aktuell geflogene Splineabschnitt in $n \in N$ Spline-Punkte $\vec{P}_n = \vec{P}(t_n)$ unterteilt. Für einen Spline-Punkt gilt $t_n = n \cdot \Delta t$, mit $\Delta t = 1/n$, so dass der Intervall $[0..1]$ weiterhin erfüllt bleibt.

Für den aktiven Spline kann, durch eine iterative Abstandsberechnung aller Spline-Punkte zur aktuellen Flugzeugposition $\vec{s}_g(t)$, der Spline-Punkt $\vec{P}(t_{min})$ ermittelt werden, der den kürzesten Abstand zur aktuellen Flugzeugposition hat. Da sich das Flugzeug aber zwischen zwei Spline-Punkten befinden kann, ist dieser Abstand noch kein gutes Maß des Abstands zur Spline selbst. Dazu wird mit der aktuellen Position $\vec{s}_g(t)$ die seitliche Bahnabweichung d, bestimmt. Über die Annahme relativ großer Bahnkrümmungen wird d, wie in Abbildung 3.5 dargestellt, über das Lot zur Spline-Tangente im Punkt $\vec{P}_{soll}$ angenähert [Sch08].

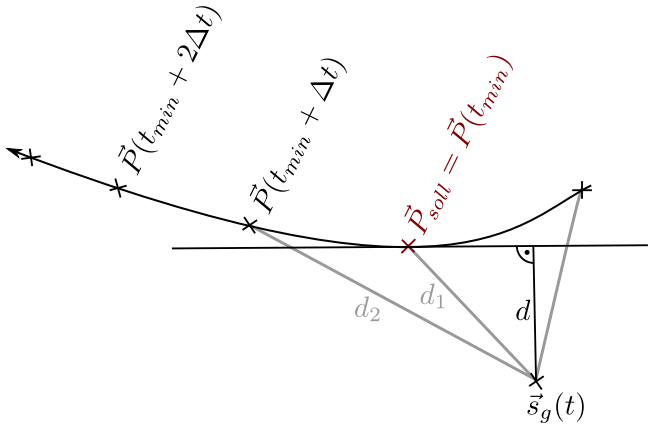


Abbildung 3.5.: Berechnung des Abstandes d, Sollbahn und aktuelle Position $\vec{s}_g(t)$.

Nach [Sch08] hat es sich am effizientesten erwiesen, zur Bahnführung den kommandierten Lagewinkel Φ_{soll} als Kombination eines Vorsteuer-Hängewinkels Φ_{ac}, Vorsteuer-Bahnazimuts χ_{ac} und der Bahnabweichung d zu bilden.

Mit der Bahngeschwindigkeit V_K, der Erdbeschleunigung g und dem lokalen Krümmungsradius des Spline-Punktes $\vec{P}(t_{min})$,

$$r = \frac{[P'_x(t_{min})^2 + P'_y(t_{min})^2]^{3/2}}{P'_x(t_{min}) \cdot P''_y(t_{min}) - P''_x(t_{min}) \cdot P'_y(t_{min})}, \tag{3.24}$$

ergibt sich der der Vorsteuer-Hängewinkel zu:

$$\Phi_{ac} = arctan\left(\frac{V_K^2}{g \cdot r}\right). \tag{3.25}$$

Der Bahnazimut lässt sich über die Ableitung der Position $\vec{P}'(t_{min})$ zwar leicht berechnen, führt aber durch die vereinfachenden Annahmen der Abstandsberechnung zu Fehlern, wodurch nach [Sch08] eine erweiterte Berechnung herangezogen wird:

$$\chi_{ac} = \frac{d_2(t_{min} + \Delta t) \cdot \chi(t_{min}) + d_1(t_{min}) \cdot \chi(t_{min} + \Delta t)}{d_1 + d_2}. \tag{3.26}$$

Dabei entspricht d_1 dem Abstand der aktuellen Flugzeugposition zur Spline-Position $\vec{P}(t_{min})$, bzw. d_2 zu $\vec{P}(t_{min} + \Delta t)$, wie in Abbildung 3.5 dargestellt.

Die Höhenregelung findet entkoppelt von der (horizontalen) Bahnregelung statt. Hierfür werden lediglich Geraden anstelle von Splines verwendet. Eine Erweiterung mit Splines ist nach dem bereits dargestellten Prinzip aber auch für die Vorgabe der Bahnhöhe denkbar [Sch08]. Der Soll-Nickwinkel Θ_{soll} bildet sich mit einem PI-Regler, aus der Differenz der gemessenen Höhe H und der gewünschten Höhe H_{soll}. Abschließend wird die Struktur der Bahnregelung in Abbildung 3.6 dargestellt. Die kommandierten Sollwinkel werden aus Sicherheitsgründen begrenzt auf $\pm 15°$ in der Nicklage und $\pm 45°$ in der Rolllage. Abweichend zu [Sch08] wird auf die Verwendung eines Fahrtreglers aus Zeitgründen verzichtet.

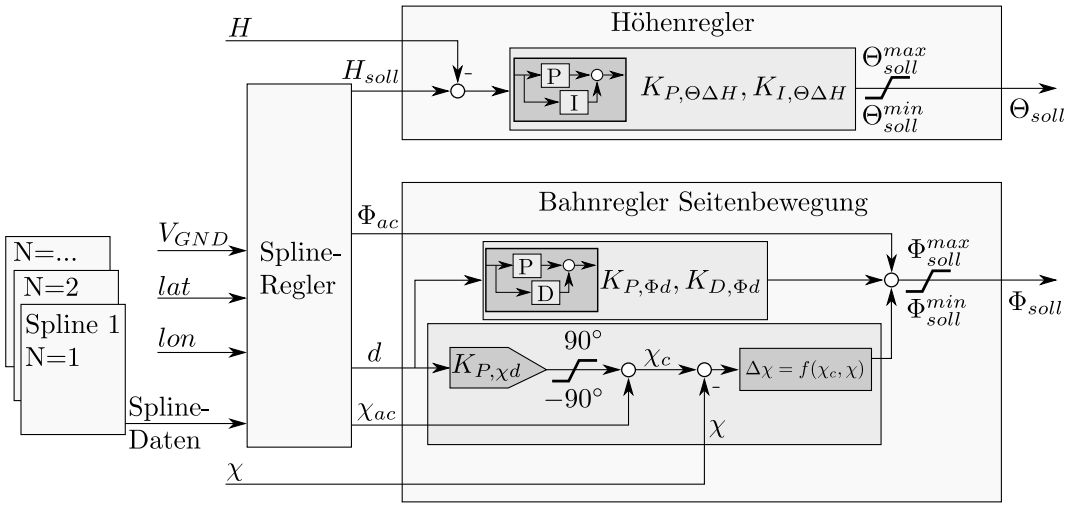


Abbildung 3.6.: Struktur des Bahnreglers, [Sch08].

3.5 Basisregler

Zur Darstellung der Problemstellung wird ein klassischer Kaskadenregler nach [BAL11] und [Sch08] verwendet. Da dieser Regler nicht Gegenstand der weiteren Untersuchungen ist, soll an dieser Stelle nur eine kurze Skizze der Ausgangssituation erfolgen. Dazu stellt Abbildung 3.7 die Regelung der Längsbewegung dar, sowie Abbildung 3.8 die der Seitenbewegung. Diese Reglerstruktur in Kombination mit dem Spline-Regler konnte bereits in diversen UAV-Missionen seine Tauglichkeit beweisen [KWR+10, KB12].

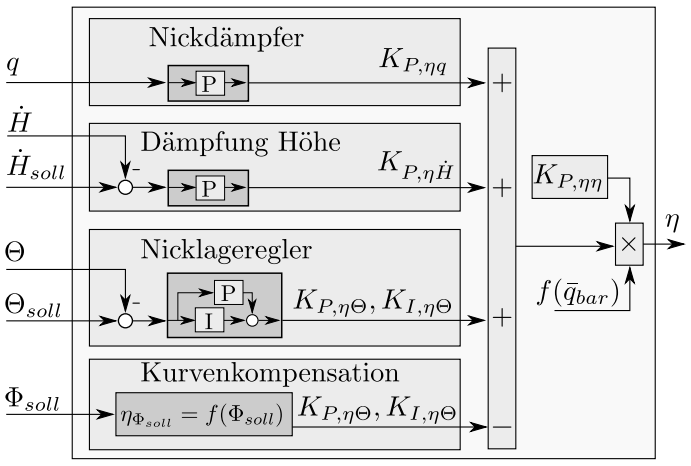


Abbildung 3.7.: Basisregler Längsbewegung, [Sch08].

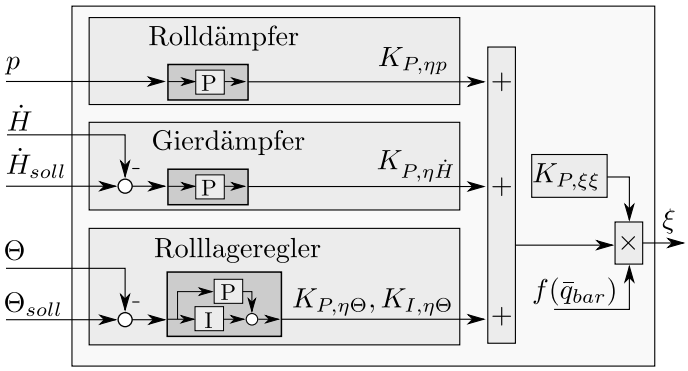


Abbildung 3.8.: Basisregler Seitenbewegung, [Sch08].

3.6 Flugversuch

Während die Modellierung der Simulation sowie die Regelung für das unbemannte Flugzeug T200 ausgelegt ist, wird in den Flugversuchen die Twinstar verwendet. Der Aufbau der Twinstar basiert im Wesentlichen aus dem Kaltschaum Elapor. Dieser Schaumstoff kann, im Falle eines Absturzes und einem damit verbundenen Bruch, mit Sekundenkleber wieder fixiert werden und bietet sich daher für Versuche besonders an. Die Flugeigenschaften der Twinstar erlauben auch einem ungeübten Hobbypiloten die Durchführung von Flugversuchen. Tatsächlich wird im Rahmen dieser Arbeit mit zwei verschiedenen Flugzeugen gearbeitet, eines zur Modellierung der Flugeigenschaften und ein anderes zur tatsächlichen Flugerprobung. Dieser Umstand muss aber keine Beeinträchtigung der erzielten Ergebnisse bedeuten, wie sich im späteren Verlauf dieser Arbeit herausstellen wird.

Im Fortgang dieser Arbeit wird immer das gleiche Experiment mit jeweils einer anderen Reglerstruktur durchgeführt. Hierbei wird vom Bahnregler die Lage $\vec{\Omega}_{soll} = [\Phi_{soll}, \Theta_{soll}, \Psi_{soll}]^T$ an den Regler kommandiert, die dieser einzuhalten hat. Nach einer gewissen Zeit wird vom Piloten mit der Fernbedienung ein simulierter Fehler im Querruder hinzu geschaltet. Dieser Fehler besteht aus dem Einfrieren des rechten Querruders bei einer Position von $\xi_{li} = 7°$. Dies entspricht 38% des Gesamtausschlags, wie in Grafik 3.9 dargestellt wird. Die Reglerparameter sind im Rahmen der Flugversuche, bis auf wenige Ausnahmen, unverändert aus der Simulation des T200 übernommen.

Um die Ausgangssituation abzubilden, wird zuerst der Kaskadenregler aus dem vorangehenden Abschnitt im Flugversuch betrachtet. Der Kaskadenregler hat sich im Rahmen dieser Arbeit in Flugversuchen mit der Twinstar als robustes und einfach

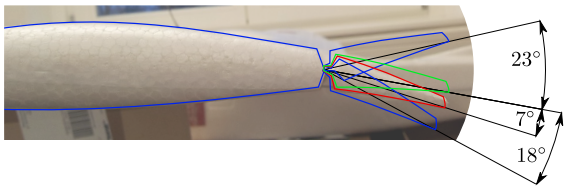


Abbildung 3.9.: Linkes Querruder; grün = ausgetrimmte Mittelposition, blau = Maximalausschlag, rot = Testeinstellung.

anzuwendendes Mittel der Bahn- und Lageregelung erwiesen. Mit der Anpassung nur eines Parameters, $K_{P,\xi\xi}$, also der Verstärkung des Querruderkommandos, wird der Kaskadenregler direkt vom T200 Modell übernommen. Während sich diese Reglerstruktur im Rahmen dieser Arbeit mit der Twinstar immer wieder als robuster Ansatz qualifiziert, zeigt dieser aber beim Hinzufügen des Querruderfehlers deutliche Schwächen, wie in der Grafik der Flugbahn 3.10 dargestellt wird. Nach dem Einschalten des Querruderfehlers hat das Flugzeug offensichtlich Schwierigkeiten der Bahn zu folgen. Eine ausgedehnte Sammlung von Flugversuchsdaten mit der Twinstar ist zudem in [Pfe14] abgebildet. Es ist nicht auszuschließen, dass mit der Wahl anderer Reglerparameter eine bessere Regelung möglich ist. Allerdings wird hier, wie sich zeigen wird, nicht die Verbesserung der Regelgüte durch Parametervariation angestrebt.

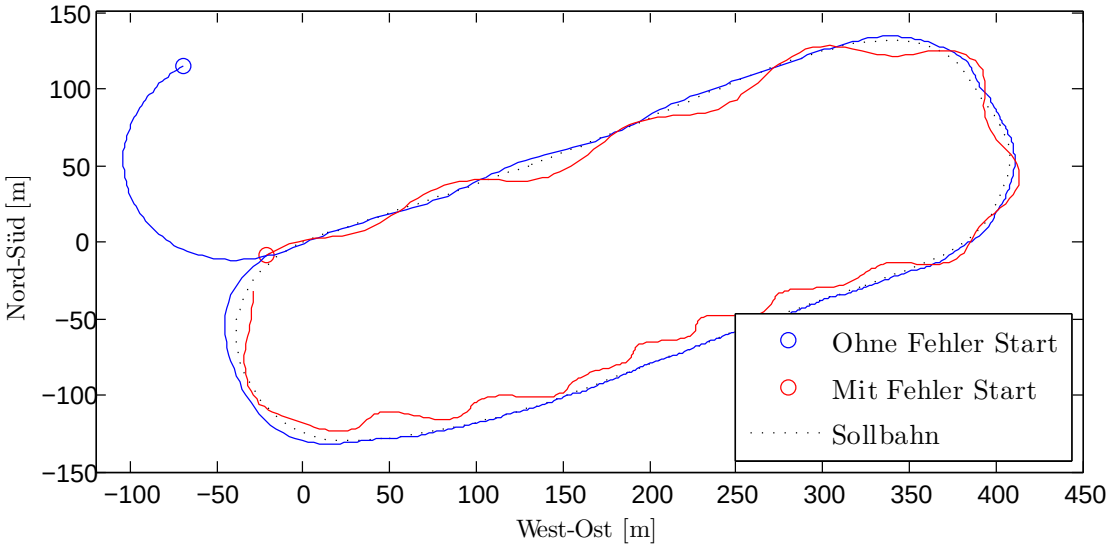


Abbildung 3.10.: Flugversuche mit Basisregler; GPS-Bahnverlauf; Blau: ohne Fehler, Rot: mit blockiertem linken Querruder.

4 Dynamische Inversion

Im letzten Abschnitt wurde bereits der Basisregler als eine Möglichkeit der Lagehaltung vorgestellt. Im Flugversuch konnte gezeigt werden, dass diese Reglerstruktur bei Systemdegradationen, z.B. dem Ausfall eines Querruders, an seine Grenzen kommt. Bevor adaptive Strukturen zur Verbesserung der Ausgangssituation vorgestellt werden können, wird mit der dynamischen Inversion die Grundlage der nichtlinearen Regelung eingeleitet.

Im Gegensatz zur linearen Regelung, wie z.B. dem Kaskadenregler, haben nichtlineare Regler Vorteile. Lineare Regler basieren auf einer Linearisierung um einen Betriebspunkt, das bedeutet für nichtlineare Systeme, dass der Linearregler unter Umständen nur innerhalb dieses Betriebspunktes zu den gewünschten Regelergebnissen führt. Im Gegensatz dazu kann, je nach System, ein nichtlinearer Regler im gesamten Zustandsraum eines Systems auslegbar sein. Das bedeutet, ein nichtlinearer Regler kann auch außerhalb eines linearisierten Bereiches und damit näher an den Grenzen des Systems betrieben werden. Eine Möglichkeit der nichtlinearen Regelung bildet die dynamische Inversion.

Vereinfacht dargestellt ist das Ziel der dynamischen Inversion, für ein nichtlineares System $\mathbf{F}(\vec{x})$ eine Rückführung $\hat{\mathbf{F}}^{-1}(\vec{x})$ zu finden, sodass durch Vorschalten der Rückführung, ein lineares Eingangs-Ausgangs-Verhalten mit der Übertragungsfunktion „1“ entsteht, wie in Bild 4.1 skizziert. Das so gewonnene lineare System kann dann idealerweise mit linearen Ansätzen geregelt werden. Die dynamische Inversion wird auch oft mit dem Begriff E/A-Linearisierung (Eingangs/Ausgangs-Linearisierung) benannt.

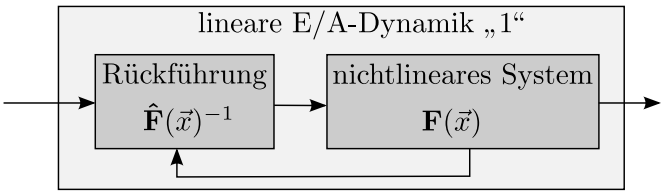


Abbildung 4.1.: Stark vereinfachte Skizze der dynamischen Inversion.

Die dynamische Inversion wird in [Isi95] allgemein behandelt, während [Kha02] und [Ada09] den Fokus auf Systeme in der Anwendung legen. Im Besonderen bildet

[Hol04, JC00] die grundlegende Inspiration für die Anwendung der dynamischen Inversion in unbemannten Flugsystemen. Im Folgenden werden die Grundlagen erläutert, um im späteren Verlauf den Bezug zur Anwendung in unbemannten Flugzeugen herzustellen.

Betrachtet wird das eingangsaffine nichtlineare MIMO-System (engl. multiple input, multiple output), d.h. das System ist nichtlinear bzgl. des Zustandsvektors $\vec{x} = [x_1, x_2, ..., x_n]^T$, aber linear im Eingangsvektor $\vec{u} = [u_1, u_2, ..., u_m]^T$:

$$\begin{aligned} \dot{\vec{x}} &= \vec{f}(\vec{x}) + \mathbf{G}(\vec{x}) \cdot \vec{u} \\ \vec{y} &= \vec{h}(\vec{x}), \end{aligned} \tag{4.1}$$

wobei $\vec{y} = [y_1, y_2, ..., y_m]^T$ dem Ausgangsvektor entspricht. Dabei wird angenommen, dass $\vec{f}$, G und $\vec{h}$ hinreichend glatt in einer Umgebung $D_x \subset R^n$, bzw. $D_u \subset R^m$ sind, d.h. mehrfach stetig differenzierbare Abbildungen darstellen. Die Abbildungen $\vec{f} : D_x \to R^n$ und $\vec{h} : D_x \to R^m$ werden auch als Vektorfelder in D bezeichnet und haben folgende Form:

$$\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), ..., f_n(\vec{x})]^T \tag{4.2}$$

sowie

$$\vec{h}(\vec{x}) = [h_1(\vec{x}), h_2(\vec{x}), ..., h_m(\vec{x})]^T. \tag{4.3}$$

Die Abbildungsmatrix $G : D_u \to R^n$ setzt sich entsprechend (4.4) aus n Vektorfeldern $\vec{g}_{n\times m}$ zusammen:

$$\mathbf{G}(\vec{x}) = \begin{bmatrix} g_{11}(\vec{x}) & \dots & g_{1m}(\vec{x}) \\ \vdots & \ddots & \vdots \\ g_{n1}(\vec{x}) & \dots & g_{nm}(\vec{x}) \end{bmatrix}. \tag{4.4}$$

Um die folgenden Erläuterungen zu vereinfachen, wird ein SISO-System (engl. single input, single output) betrachtet, wobei diese einfach auf MIMO-Systeme erweiterbar sind [Ada09]:

$$\begin{aligned} \dot{\vec{x}} &= \vec{f}(\vec{x}) + \vec{g}(\vec{x}) \cdot u \\ y &= h(\vec{x}). \end{aligned} \tag{4.5}$$

Da sich die Darstellung (4.5) für den angestrebten Reglerentwurf als unvorteilhaft erwiesen hat, wird das System zuerst in die eingangsnormalisierte Normalform, auch eingangsnormalisierte Byrnes-Isidory-Normalform genannt [BI84], transformiert. Ziel dieser Transformation ist die Unterteilung des Systems in eine Integratorkette mit dem relativen Grad r und der sogenannten internen Dynamik. Die im Blockdiagramm 4.2 skizzierte Transformation soll im folgenden Verlauf hergeleitet werden.

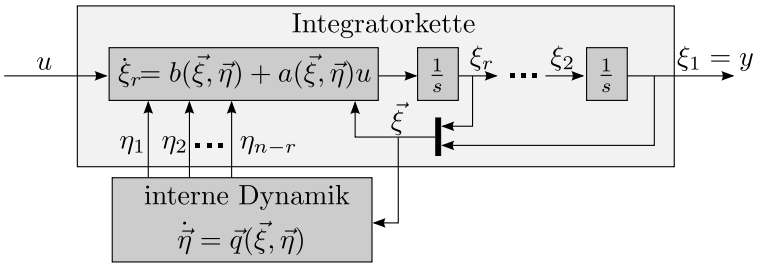


Abbildung 4.2.: Trennung zwischen Integratorkette und interner Dynamik (vgl. Gleichung (4.12)).

Der relative Grad r entspricht dabei der Anzahl von Integratoren, die nötig sind, um den Zusammenhang zwischen dem Eingang u und dem Ausgang y herzustellen. Dagegen ist die interne Dynamik nicht vom Eingang u steuerbar und kann mit den Nullstellen eines linearen Systems gedeutet werden [Eng95]. Um die erwähnte Normalform zu erreichen, muss zuerst der relative Grad r bestimmt werden.

4.1 Der relative Grad

Der relative Grad in einer Umgebung von $\vec{x} = \vec{x}_0$ ergibt sich aus der kleinsten natürlichen Zahl r, bei der für das System (4.5) gilt (weitere Herleitung der Lie-Algebra siehe [BI84]):

$$L_g L_f^{r-1} h(\vec{x}) \neq 0. \tag{4.6}$$

Darin enthalten sind mit $L_g L_f$, die sogenannten Lie-Ableitungen, die den Gesetzmäßigkeiten aus folgendem Beispiel folgen:

$$
\begin{aligned}
L_g L_f^{r-1} h(\vec{x}) &= \frac{\partial (L_f^{r-1} h)}{\partial \vec{x}} g(\vec{x}) \\
L_f^{r-1} h(\vec{x}) = L_f L_f^{r-2} h(\vec{x}) \qquad &= \frac{\partial (L_f^{r-2} h)}{\partial \vec{x}} f(\vec{x}) \\
&\vdots \\
L_f^2 h(\vec{x}) = L_f L_f h(\vec{x}) \qquad &= \frac{\partial (L_f h)}{\partial \vec{x}} f(\vec{x}) \\
L_f h(\vec{x}) &= \frac{\partial (h)}{\partial \vec{x}} f(\vec{x}) \\
L_f^0 h(\vec{x}) &= h(\vec{x}).
\end{aligned}
\tag{4.7}
$$

Betrachtet man nun die Erweiterung der ersten Ableitung $\dot{y}$ des Ausgangs, in Kombination mit Gleichung (4.5) sowie (4.7),

$$
\dot{y} = \frac{\partial y}{\partial x} \frac{\partial x}{\partial t} = \frac{\partial h}{\partial x} [\vec{f}(\vec{x}) + \vec{g}(\vec{x}) \cdot u] = L_f \cdot h(\vec{x}) + L_g \cdot h(\vec{x}) \cdot u, \tag{4.8}
$$

so bleibt zu überprüfen, ob Gleichung (4.8) in einer Umgebung $\vec{x} = \vec{x}_0$ explizit vom Eingang u abhängig ist. Ist dies nicht der Fall, so gilt $L_g \cdot h(\vec{x}) = 0$, bzw. Gleichung (4.6) mit $r = 1$ als nicht erfüllt und das Vorgehen ist bis zur r-ten Ableitung $y^{(r)}$ fortzuführen:

$$
\begin{aligned}
\ddot{y} &= L_f^2 h(\vec{x}) + L_g L_f \cdot h(\vec{x}) \cdot u \\
&\vdots \\
y^{(r)} &= L_f^r \cdot h(\vec{x}) + L_g L_f^{r-1} \cdot h(\vec{x}) \cdot u.
\end{aligned}
\tag{4.9}
$$

So wird klar, dass der Formalismus (4.6) dann erfüllt ist, wenn die r-te Ableitung des Ausgangs $y^{(r)}$ zum ersten mal explizit vom Eingang u abhängt. Mit diesem Vorgehen wurde somit der relative Grad r eindeutig bestimmt.

4.2 Byrnes-Isidory-Normalform

Mit der Bestimmung des relativen Grades r wurde zwar eine Kennzahl zwischen E/A-Dynamik und interner Dynamik gefunden, für die Bestimmung der Transformation zur exakten E/A-Linearisierung müssen diese zwei Dynamiken des Systems aber in Gleichung (4.5), wie bereits erwähnt, formal getrennt werden. Dies geschieht durch die sogenannte Normalform, die auch als Byrnes-Isidory-Normalform (BINF) bekannt ist [BI84]. Um diese zu bestimmen, werden hierfür mit

$$\vec{z} = \Phi(\vec{x}) = \begin{bmatrix} \Phi_\xi(\vec{x}) \\ \Phi_\eta(\vec{x}) \end{bmatrix} = \begin{bmatrix} \vec{\xi} \\ \vec{\eta} \end{bmatrix} \tag{4.10}$$

neue Koordinaten eingeführt, wobei $\vec{\xi} = [\xi_1, \xi_2, ..., \xi_r]$ und $\vec{\eta} = [\eta_1, \eta_2, ..., \eta_{n-r}]$ entsprechen. Die ersten $\vec{\xi}$ Koordinaten ergeben sich durch Ableiten des Ausgangs y, wie bereits mit Gleichung (4.8) und (4.9) beschrieben. In Anlehnung an [Ada09, Isi95, Hol04] entspricht der Index an ξ_r dem relativen Grad $r - 1$:

$$\begin{aligned} \xi_1 &= \Phi_1(\vec{x}) = y = L_f^0 \cdot h(\vec{x}) = h(\vec{x}) \\ \xi_2 &= \Phi_2(\vec{x}) = \dot{\xi}_1 = \dot{y} = L_f^1 \cdot h(\vec{x}) = L_f \cdot h(\vec{x}) \\ &\vdots \\ \xi_{r-1} &= \Phi_{r-1}(\vec{x}) = \dot{\xi}_{r-2} = y^{(r-2)} = L_f^{r-2} \cdot h(\vec{x}) \\ \xi_r &= \Phi_r(\vec{x}) = \dot{\xi}_{r-1} = y^{(r-1)} = L_f^{r-1} \cdot h(\vec{x}). \end{aligned} \tag{4.11}$$

Offensichtlich entfallen alle Terme $L_g L_f^{k-1} \cdot h(\vec{x})$ aus (4.9) mit $k < r$, da wie inzwischen berechnet, der Eingang u erst ab dem r-ten Term sichtbar wird. Mit der Transformationsvorschrift $\vec{x} = \Phi^{-1}(\vec{z})$ kann nun, zur Quantifizierung der Systemdynamik, in 4.11 die Normalform aufgestellt werden:

$$\begin{aligned} \dot{\xi}_1 &= \xi_2 \\ \dot{\xi}_2 &= \xi_3 \\ &\vdots \\ \dot{\xi}_{r-1} &= \xi_r \\ \dot{\xi}_r &= b(\vec{\xi}, \vec{\eta}) + a(\vec{\xi}, \vec{\eta}) \cdot u \\ \dot{\vec{\eta}} &= \vec{q}(\vec{\xi}, \vec{\eta}) + \vec{p}(\vec{\xi}, \vec{\eta}) \cdot u, \end{aligned} \tag{4.12}$$

mit der Substitution

$$b(\vec{\xi}, \vec{\eta}) = L_f^r \cdot h(\Phi^{-1}(\vec{z})), \qquad a(\vec{\xi}, \vec{\eta}) = L_g L_f^{r-1} \cdot h(\Phi^{-1}(\vec{z})) \tag{4.13}$$

sowie

$$q(\vec{\xi}, \vec{\eta}) = L_f \cdot \Phi_\eta(\Phi^{-1}(\vec{z})), \qquad p(\vec{\xi}, \vec{\eta}) = L_g \cdot \Phi_\eta(\Phi^{-1}(\vec{z})). \tag{4.14}$$

Der Ausgang entspricht dabei:

$$y = \xi_1. \tag{4.15}$$

Deutlich erkennbar ist hierbei inzwischen die am Anfang erwähnte E/A-Dynamik sowie die interne Dynamik, wie in Darstellung 4.2 gezeigt. Allerdings hängt die Normalform noch vom Eingang u ab, entspricht also noch nicht der gewünschten eingangsnormalisierten Normalform. Um die dargestellte Form aus Grafik 4.2 zu erreichen, müssen die $\vec{\eta}$ Koordinaten ohne den Eingang u aufgestellt werden. In [Isi95] wurde gezeigt, dass für ein System mit einem relativen Grad r, immer $n-r$ Funktionen in einer Umgebung von $\vec{x} = \vec{x}_0$ mit der Nebenbedingung

$$L_g \Phi_\eta(\vec{x}) = 0 \tag{4.16}$$

und der Eigenschaft der linearen Unabhängigkeit der Zeilenvektoren von $\Phi(\vec{x})$, also

$$rg\left[\frac{\partial \Phi}{\partial \vec{x}}(\vec{x}_0)\right] = n-1, \tag{4.17}$$

existieren. Mit diesen Bedingungen können genügend Gleichungen aufgestellt werden, um die fehlenden $\Phi_\eta(\vec{x}) = \vec{\eta}$ Koordinaten zu bilden, die um $\vec{x} = \vec{x}_0$ in $\Phi(x)$ einen Diffeomorphismus bilden. Damit ist die Invertierbarkeit von $\vec{z} = \Phi(\vec{x})$ sichergestellt, womit eine Rücktransformation $\vec{x} = \Phi(\vec{z})^{-1}$ möglich ist. Angemerkt sei hierbei, dass unter bestimmten Bedingungen, mehrere Lösungen für $\vec{\eta}$ möglich sind. Damit bildet sich die eingangsnormalisierte Normalform, bzw. eingangsnormalisierte Byrnes-Isidory-Normalform, die in Vektornotation folgende Form annimmt:

$$\dot{\vec{\xi}} = \begin{bmatrix} \xi_2 \\ \vdots \\ \xi_r \\ a(\vec{\xi},\vec{\eta}) + b(\vec{\xi},\vec{\eta}) \cdot u \end{bmatrix}, \dot{\vec{\eta}} \ = \vec{q}(\vec{\xi},\vec{\eta}). \tag{4.18}$$

Der Übersicht halber sei angemerkt, dass die Substitutionen (4.13) auch in den ursprünglichen Koordinaten dargestellt werden können. Mit der Transformationsvorschrift $\vec{x} = \Phi(\vec{z})^{-1} = \Phi(\vec{\xi},\vec{\eta})^{-1}$ gilt:

$$a(\vec{\xi},\vec{\eta}) = a(\vec{x}), \quad b(\vec{\xi},\vec{\eta}) = b(\vec{x}), \quad q(\vec{\xi},\vec{\eta}) = p(\vec{x}), \quad p(\vec{\xi},\vec{\eta}) = p(\vec{x}), \tag{4.19}$$

Angelehnt an [Isi95, SW91, Hol04] kann die Byrnes-Isidory-Normalform auch einfach auf MIMO-Systeme der Form in Gleichung (4.1) erweitert werden. Analog zur Berechnung des relativen Grades r im Eingrößensystem kann, mit dem Vorgehen aus Gleichung (4.8) und (4.9), der relative Grad r für das Mehrgrößensystem bestimmt

werden. Mit dem Eingangsvektor $\vec{u}_{m\times 1}$ und dem Ausgangsvektor $\vec{y}_{m\times 1}$ berechnet sich ein Element r_i vom relativen Grad-Vektor $\vec{r} = [r_1, r_2, ..., r_m]$ über das kleinste r_i, bei dem für eine Ableitung des Ausgangs y_i, zum erstem Mal ein Element des Eingangsvektors $\vec{u}$ sichtbar wird. So ergibt sich folgender Formalismus:

$$\underbrace{\begin{bmatrix} y_1^{(r_1)} \\ y_2^{(r_2)} \\ \vdots \\ y_m^{(r_m)} \end{bmatrix}}_{[y_i{}^{(r_i)}]_{m\times 1}} = \underbrace{\begin{bmatrix} L_f^{r_1} \cdot h_1(\vec{x}) \\ L_f^{r_2} \cdot h_2(\vec{x}) \\ \vdots \\ L_f^{r_m} \cdot h_m(\vec{x}) \end{bmatrix}}_{\mathbf{b}(\vec{x})} + \underbrace{\begin{bmatrix} L_{g1} \cdot L_f^{r_1-1} \cdot h_1(\vec{x}) & \dots & L_{gm} \cdot L_f^{r_1-1} \cdot h_1(\vec{x}) \\ L_{g1} \cdot L_f^{r_2-1} \cdot h_2(\vec{x}) & \dots & L_{gm} \cdot L_f^{r_2-1} \cdot h_2(\vec{x}) \\ \vdots & \ddots & \vdots \\ L_{g1} \cdot L_f^{r_m-1} \cdot h_m(\vec{x}) & \dots & L_{gm} \cdot L_f^{r_m-1} \cdot h_m(\vec{x}) \end{bmatrix}}_{\mathbf{A}(\vec{x})} \underbrace{\begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix}}_{\vec{u}}. \tag{4.20}$$

Dabei ist darauf zu achten, ob die $m \times m$ Matrix $\mathbf{A}(\vec{x})$ invertierbar ist, da sonst per Definition kein relativer Grad r_i existiert. Diese Matrix $\mathbf{A}(\vec{x})$ wird auch Entkopplungsmatrix genannt. Der Vektor der Ausgangsableitungen $\vec{y}^{(r_m)}$ kann, analog zu den Substitutionen (4.13), auch durch

$$[y_i^{(r_i)}]_{m\times 1} = \mathbf{b}(\vec{x}) + \mathbf{A}(\vec{x}) \cdot \vec{u} = \mathbf{b}(\vec{\xi}, \vec{\eta}) + \mathbf{A}(\vec{\xi}, \vec{\eta}) \cdot \vec{u} \tag{4.21}$$

dargestellt werden. Die Elemente des transformierten Systems entsprechen hierbei $\xi_r^i = y_i^{(r_m-1)}$, sodass für den Ausgang des Systems (4.1) gilt:

$$\xi_1^i = y_i. \tag{4.22}$$

Die formale Beschreibung des transformierten Mehrgrößensystems erfolgt durch die Aneinanderreihung der transformierten Elemente, wodurch der Gesamtzustandsvektor die Form

$$\vec{\xi} = [\xi_1^1 \dots \xi_{r_1}^1, \xi_1^2 \dots \xi_{r_2}^2, ..., \xi_1^m \dots \xi_{r_m}^m]^T \tag{4.23}$$

annimmt. Definiert man die Summe der Elemente der relativen Grade als

$$r_{sum} = \sum_{i=1}^{m} r_i, \tag{4.24}$$

so können mit den Bedingungen (4.16) und (4.17), analog $n - r_{sum}$ Gleichungen für $\vec{\eta}$ gefunden werden, um in $\Phi(\vec{x})$ einen Diffeomorphismus zu bilden. Dadurch ist auch für den MIMO Fall die Existenz der Inversen $\Phi^{-1}(\vec{x})$ garantiert.

4.3 Linearisierende Zustandsrückführung

Mit der Bestimmung der eingangsnormalisierten Byrnes-Isidory-Normalform konnte eine Entkopplung von E/A-Dynamik und interner Dynamik vollzogen werden. Mit dem ursprünglichen Ziel der E/A-Linearisierung wird nun eine Zustandsrückführung vorgestellt, die mit dem nichtlinearen System (4.1) ein lineares E/A-Verhalten bereitstellt. Angelehnt an die Nomenklatur von [Hol04, Krü12], kann nun mit der Einführung der Substitutionen $\vec{\alpha}(\vec{x})$ und $\vec{\beta}(\vec{x})$ die linearisierende Zustandsrückführung implementiert werden:

$$\vec{u} = \vec{\alpha}(\vec{x}) + \vec{\beta}(\vec{x}) \cdot \nu = \vec{\alpha}(\vec{\xi}, \vec{\eta}) + \vec{\beta}(\vec{\xi}, \vec{\eta}) \cdot \vec{\nu}, \tag{4.25}$$

wobei $\vec{\nu} = [\nu_1, \nu_2, ..., \nu_m]^T$ der sogenannten Ersatzregelgröße entspricht. Durch geschickte Wahl von $\vec{\alpha}(\vec{x})$ und $\vec{\beta}(\vec{x})$,

$$\vec{\alpha}(\vec{x}) = -\mathbf{A}^{-1}(\vec{x}) \cdot \mathbf{b}(\vec{x}), \qquad \vec{\beta}(\vec{x}) = \mathbf{A}^{-1}(\vec{x}), \tag{4.26}$$

ergibt sich die tatsächliche Steuergröße zu:

$$\vec{u} = \mathbf{A}^{-1}(\vec{x})[\vec{\nu} - \mathbf{b}(\vec{x})]. \tag{4.27}$$

Eingesetzt in (4.21), wird die r_i-te Ableitung eines Elements des Ausgangsvektors $[y_i^{(r_i)}]_{m\times 1}$ sofort linear bezüglich der Ersatzregelgröße $\vec{\nu}$:

$$\vec{\nu} = [y_i^{(r_i)}]_{m\times 1}. \tag{4.28}$$

Durch diese Rückführung wurde also die zu Anfang des Kapitels gesuchte E/A-Linearisierung bewerkstelligt.

Zum Zwischenfazit soll eine Betrachtung der Rückführung in Abbildung 4.3 am SISO-Fall stattfinden. Zu sehen ist die E/A-Dynamik $\dot{\xi}_r$, auch externe Dynamik genannt, mit der Integratorkette und der vorgeschalteten Rückführung. Die restliche $\vec{\eta}$ Dynamik ist weder steuerbar noch beobachtbar und wird daher interne Dynamik genannt. Die interne Dynamik bedarf daher weiterer Betrachtung, die im nächsten Abschnitt erfolgen soll. Besitzt das System einen relativen Grad $r = n$, bzw. im MIMO-Fall $r_{Ges} = n$, dann existiert keine interne Dynamik und man spricht nicht mehr von der exakten E/A-Linearisierung, sondern von der exakten Zustandslinearisierung. Die Ersatzregelgröße ν ist linear „1“ bezüglich des Ausgangs $\dot{\xi}_r$ der

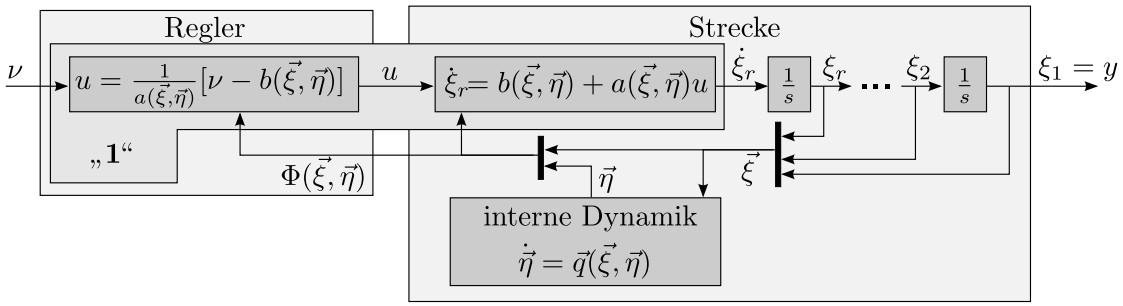


Abbildung 4.3.: Darstellung des E/A-linearisierten SISO Systems.

externen Dynamik und damit der r-ten Ableitung des Ausgang y. Offensichtlich wird der Ausgang y über r Integratoren mit $\dot{\xi}_r$ verbunden, woraus folgt, dass die Ersatzregelgröße ν nicht direkt den Ausgang y steuert und daher auch als Pseudosteuergröße bezeichnet wird. Um eine Steuerung des direkten Ausgangs y zu erwirken, wird ein Referenzsignal y_{ref} benötigt, wie später im zugehörigen Abschnitt 4.5 erörtert wird.

Abschließend wird in Grafik 4.4 ersichtlich, dass neben der E/A-Linearisierung zusätzlich eine Entkopplung des E/A-Verhaltens stattfindet. Wenn mit einem Element

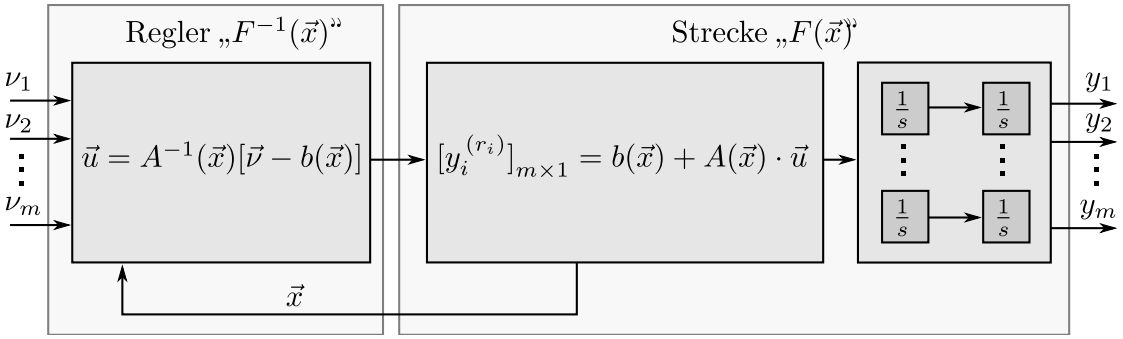


Abbildung 4.4.: Darstellung des E/A-linearisierten MIMO Systems.

u_i des Eingangsvektors vom nichtlinearen System (4.1) mehrere Ausgänge y_i beeinflusst wurden, so wird jetzt mit dem neuen Eingangselement ν_i immer nur der entsprechende Anteil des Ausgangs y_i angeregt. Wie bereits erwähnt, hat die E/A-Linearisierung zur Folge, dass zwischen dem neuen Eingang, der Pseudosteuergröße $\vec{\nu}$ und dem Ausgang $[y_i^{(r_i)}]_{m\times 1}$ das lineare E/A-Verhalten „1“ entsteht. Diese Neuerung kann für die Darstellung eines vereinfachten Ersatzsystems genutzt werden, wie in Grafik 4.5 abgebildet.

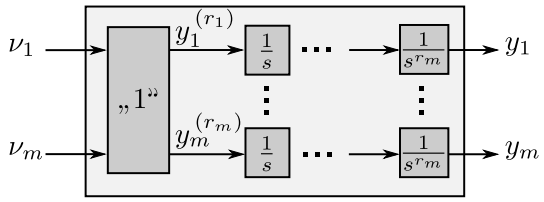


Abbildung 4.5.: Darstellung des vereinfachten Ersatzsystems.

4.4 Interne Dynamik und Nulldynamik

Im letzten Abschnitt wurde ein nichtlineares System in die externe und die interne Dynamik (vgl. Gleichung (4.18)) aufgeteilt. Mit der E/A-Linearisierung wurde ein Weg gefunden, ein lineares E/A-Verhalten zu erreichen. Zwar ist die interne Dynamik im Rahmen der eingangsnormalisierten Byrnes-Isidory-Normalform weder steuerbar noch beobachtbar, betrachtet man aber den tatsächlichen Eingang in das System (4.1):

$$\vec{u} = \mathbf{A}(\vec{\xi}, \vec{\eta})^{-1}[\vec{\nu} - \mathbf{b}(\vec{\xi}, \vec{\eta})], \tag{4.29}$$

so ist deutlich sichtbar, dass die interne Dynamik durchaus einen Einfluss auf das System (4.1) haben kann. Diese Tatsache begründet eine weitergehende Betrachtung der internen Dynamik.

Mit dem Ziel einer Folgeregelung gilt für die Ausgangsgrößen $\vec{y}$, deren Ableitungen $[y_i^{(r_i)}]_{m\times 1}$ und damit für die Zustände $\vec{\xi}$ nach der Implementierung der exakten E/A-Linearisierung mit einem Referenzverlauf $\vec{y}_{ref}$:

$$\begin{aligned}\vec{\xi}(t) = \vec{\xi}_{ref}(t) &= [\xi^1_{1ref} \dots \xi^1_{r_1 ref}, \xi^2_{1ref} \dots \xi^2_{r_2 ref}, ..., \xi^m_{1ref} \dots \xi^m_{r_m ref}]^T \\ &= [y_{1ref}, ..., y^{r_1-1}_{1ref}, y_{2ref}, ..., y^{r_2-1}_{2ref}, ..., y_{mref}, ..., y^{r_m-1}_{mref}]^T.\end{aligned} \tag{4.30}$$

Mit der Anfangsbedingung $\vec{\eta}(t) = \vec{\eta}_0$ entspricht damit die interne Dynamik in eingangsnormalisierter Byrnes-Isidory-Normalform:

$$\dot{\vec{\eta}} = \vec{q}(\vec{\xi}_{ref}(t), \vec{\eta}(t)). \tag{4.31}$$

Damit ist es unabdingbar, dass die interne Dynamik ein stabiles oder mindestens beschränktes Verhalten besitzt, um die Regelungsaufgabe in adäquater Regelgüte zu erfüllen. Auch wenn die interne Dynamik auf das E/A-Verhalten keine Auswirkung hat, so könnte ein unkontrolliertes Ansteigen der internen Dynamik ein physikali-

sches System an seine Grenzen bringen und zerstören. Ein weiteres Manko ergibt sich durch die nicht gegebene Beobachtbarkeit. Die Rückführung (4.29) bedingt mindestens die Kenntnis über die Beschränktheit der internen Dynamik. So kann der Zustand der internen Dynamik für die linearisierende Rückführung z.B. als konstant angenommen werden:

$$\vec{u} = \mathbf{A}(\vec{\xi}(t), \vec{\eta}_c)^{-1}[\vec{\nu} - \mathbf{b}(\vec{\xi}(t), \vec{\eta}_c)]. \tag{4.32}$$

Im Sinne der Untersuchung der internen Dynamik, kann die sogenannte Nulldynamik herangezogen werden [Krü12, Hol04, Isi95]. Wählt man den Ausgang $y_{ref} = 0$ als Referenzgröße, folgt für dessen Ableitungen und damit für die Zustände $\vec{\xi}(t) = 0, \forall t$. Damit wird die interne Dynamik zur sogenannten Nulldynamik:

$$\dot{\vec{\eta}} = \vec{q}(0, \vec{\eta}(t)). \tag{4.33}$$

Da die Nulldynamik von den Ausgangsgrößen $[y_i^{(r_i)}]_{m\times 1}$ entkoppelt ist, wird eine Stabilitätsbetrachtung deutlich vereinfacht. Damit ist eine Übertragung des Prinzips der Minimalphasigkeit aus dem Gebiet der linearen Systeme auf nichtlineare Systeme möglich. Liegt eine asymptotisch stabile Nulldynamik vor, kann von einem asymptotisch minimalphasigen System gesprochen werden. Ist dazu der Ursprung $\vec{\eta} = 0$ der Nulldynamik eine asymptotisch stabile Ruhelage, folgt, dass die Nulldynamik mindestens eine lokale Eingangsstabilität besitzt. Diese Erkenntnis ist keine notwendige, aber eine hinreichende Bedingung für die Stabilität des E/A-linearisierten Gesamtsystems, die für eine Folgeregelung benötigt wird [Hol04].

4.5 Referenzmodell

Mit der E/A-Linearisierung steht nun ein lineares System zur Verfügung. In erster Instanz könnte diesem System jede beliebige Dynamik aufgesetzt werden, aber in der Praxis sind der Wunschdynamik Grenzen gesetzt. Vor allem die Stellaktuatorik, physikalische Belastungsgrenzen und auch die Messdynamik setzen die Schranken. Gleichzeitig wird mit der Pseudosteuergröße $\vec{\nu}$ nicht direkt der Ausgangsvektor $\vec{y}$, sondern dessen r-te Ableitung $[y_i^{(r_i)}]_{m\times 1}$ direkt beeinflusst. Damit wird die Forderung eines Modells plausibel, das sowohl die Referenzgröße $\vec{y}_{ref}$ wie auch r-te Ableitung $[y_{ref,i}^{(r_i)}]_{m\times 1}$ generiert. Gleichzeitig wird eine Möglichkeit benötigt, den Sollwertverlauf zu begrenzen und damit eine Wunschdynamik zu implementieren.

Zur Bestimmung der Referenzverläufe werden in der Regel sogenannte lineare Referenzmodelle verwendet, womit die Solldynamik einfach und strukturiert vorgegeben werden kann. Beispielhaft am Bild 4.6 wird ein Referenzmodell $G_{ref}(s)$ zweiter Ord-

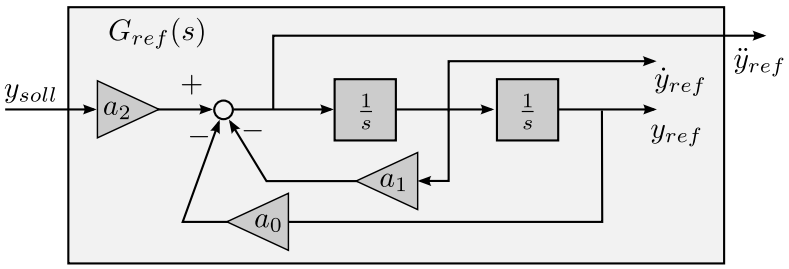


Abbildung 4.6.: Beispiel für ein Referenzmodell zweiter Ordnung.

nung dargestellt. Mit den Größen $[a_0, a_1, a_2]$ kann hierbei eine an die Grenzen des Systems angepasste Dynamik entworfen werden. Verallgemeinert auf ein System (4.5) mit dem relativen Grad r, wird das Referenzsignal $y_{ref}^{(r)}$ bestimmt mit

$$y_{ref}^{(r)} = -\sum_{i=1}^{r-1} a_i \cdot y^{(i)} + a_0 \cdot y_{soll}. \tag{4.34}$$

Dieses Vorgehen kann für den MIMO-Fall erweitert werden, dabei entspricht die Pseudosteuergröße $\vec{\nu}$ immer der höchsten Ableitung aus dem Referenzsignalgenerator:

$$\vec{\nu} = [y_{ref,i}^{(r_i)}]_{m\times 1}. \tag{4.35}$$

Anschaulich wird dies mit der Erweiterung des Ersatzsystems aus der Darstellung 4.5 in der erweiterten Abbildung 4.7 skizziert.

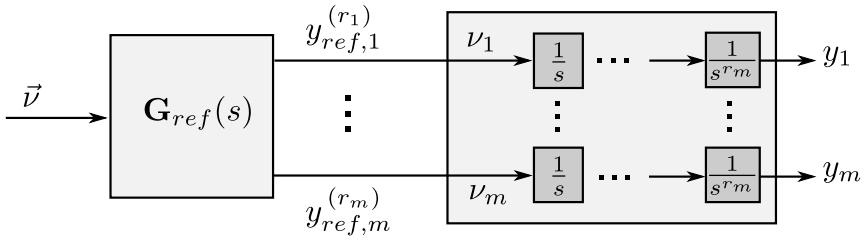


Abbildung 4.7.: Darstellung eines Referenzmodells $G_{ref}(s)$ mit vereinfachtem Ersatzsystem.

4.6 Inversionsfehler und Fehlerdynamik

Die Theorie der E/A-Linearisierung in der bisherigen Ausführung basiert auf einem perfekt invertierten System, wofür ein perfektes Modell angenommen wurde. Dass dies nicht der Realität entsprechen kann, ist mit Messfehlern, Modellierungsfehlern, Aktuatorbegrenzungen usw. zu begründen. So führt z.B. schon eine leichte Abweichung im Flugzeuggewicht zu einer Abweichung zwischen Modell und Realität. Tatsächlich kann ein reales System immer nur als Näherung in den Modellgleichungen abgebildet werden. Dazu soll im weiteren Verlauf das abweichende Modell mit dem Dach „ˆ“ in den Systemgleichungen, angelehnt an [Krü12, Hol04], dargestellt werden:

$$\begin{aligned} \dot{\vec{x}} &= \hat{\vec{f}}(\vec{x}) + \hat{\mathbf{G}}(\vec{x}) \cdot u \\ \hat{\vec{y}} &= \hat{\vec{h}}(\vec{x}). \end{aligned} \tag{4.36}$$

Das Ziel der folgenden Ausführungen ist eine Quantifizierung des Einflusses eines fehlerhaften Modells auf die Systemdynamik. Dazu wird zuerst der Einfluss des Fehlers in der eingangsnormalisierten Byrnes-Isidory-Normalform ermittelt. Parallel zu Gleichung (4.21) folgt für den abgeleiteten Ausgang $[\hat{y}_i^{(r_i)}]_{m\times 1}$ des Modells (vgl. Gl. (4.36)), dass eine Abweichung dieses Modells gegenüber dem realen System (vgl. Gl. (4.1)) zu einer Abweichung $\vec{\Delta}$ in dessen Normalform führt:

$$\begin{aligned} [\hat{y}_i^{(r_i)}]_{m\times 1} &= \hat{\mathbf{b}}(\vec{x}) + \hat{\mathbf{A}}(\vec{x}) \cdot \vec{u} \\ &= \mathbf{b}(\vec{x}) + \mathbf{A}(\vec{x}) \cdot \vec{u} + \vec{\Delta}. \end{aligned} \tag{4.37}$$

Die Abweichung $\vec{\Delta}$ wird somit auch als Inversionsfehler bezeichnet. Wird die Gleichung (4.37) analog zur exakten E/A-Linearisierung nach der tatsächlichen Steuergröße $\vec{u}$ umgestellt, folgt für diese, jeweils abhängig vom falsch modellierten (4.36) bzw. exakten System (4.1):

$$\begin{aligned} \vec{u} &= \hat{\mathbf{A}}^{-1}(\vec{x})[\vec{\nu} - \hat{\mathbf{b}}(\vec{x})] \\ &= \mathbf{A}^{-1}(\vec{x})[\vec{\nu} - \mathbf{b}(\vec{x}) + \vec{\Delta}]. \end{aligned} \tag{4.38}$$

Anschaulich bedeutet dies, dass der Fehler $\vec{\Delta}$ fehlerhafte Steuergrößen zur E/A-Linearisierung erstellt. Um den Einfluss des Fehlers zwischen realem und modelliertem System genauer zu verstehen, wird nun die linearisierende Rückführung betrachtet. Setzt man die Gleichung (4.38) in die entsprechende Zeile der Byrnes-Isidory-

Normalform des tatsächlichen Systems (4.21), folgt für den Ausgang des Systems mit der fehlerhaften E/A-Linearisierung:

$$\begin{aligned}[y_i^{(r_i)}]_{m\times 1} &= \mathbf{b}(\vec{x}) + \mathbf{A}(\vec{x}) \cdot \vec{u} \\ &= \mathbf{b}(\vec{x}) + \mathbf{A}(\vec{x}) \cdot \hat{\mathbf{A}}^{-1}(\vec{x})[\vec{\nu} - \hat{\mathbf{b}}(\vec{x})] \\ &= \mathbf{b}(\vec{x}) + \mathbf{A}(\vec{x}) \cdot \mathbf{A}^{-1}(\vec{x})[\vec{\nu} - \mathbf{b}(\vec{x}) + \vec{\Delta}].\end{aligned} \tag{4.39}$$

Für die Realisierung des Reglers wird weiter die Implementierung einer Folgeregelung angestrebt. Dazu kann auf das System wieder ein Referenzverlauf angewendet werden, woraus $\vec{\nu} = [y_{ref,i}^{(r_i)}]_{m\times 1}$ folgt, was eingesetzt in Gleichung (4.39) (vgl. auch Gl. (4.28)), den Inversionsfehler $\vec{\Delta}$ in eine anschauliche Darstellung führt:

$$[y_i^{(r_i)}]_{m\times 1} = \vec{\nu} + \vec{\Delta} \quad = [y_{ref,i}^{(r_i)}]_{m\times 1} + \vec{\Delta}, \tag{4.40}$$

bzw.:

$$\vec{\Delta} = [y_i^{(r_i)}]_{m\times 1} - [y_{ref,i}^{(r_i)}]_{m\times 1}. \tag{4.41}$$

Zur besseren Anschauung kann nun das fehlerhaft E/A-linearisierte System mit dem um den Inversionsfehler $\vec{\Delta}$ erweiterten Ersatzsystem aus Abbildung 4.5, in Abbildung 4.8 dargestellt werden. Diese Interpretation enthält eine wichtige Aussage: Offensichtlich wird der Ausgang vom gewünschten Referenzverlauf abweichen, wenn die Modellierung der Systemdynamik nicht exakt erfolgt ist. Der entstandene Fehler wird zudem mit der Integratorkette fortgepflanzt. Werden an dieser Stelle keine Maßnahmen getroffen, kann dies fatale Folgen für die Stabilität des gesamten Regleraufbaus haben. Eine Möglichkeit, dem entgegenzuwirken, ist eine Rückführung der Fehler. Dazu wird eine andere Betrachtung des Fehlers Δ eingeführt und erlaubt so im späteren Verlauf dieses Abschnittes eine Darstellung der Fehlerdynamik und darüber hinaus eine Einflussnahme auf die Fehlerdynamik.

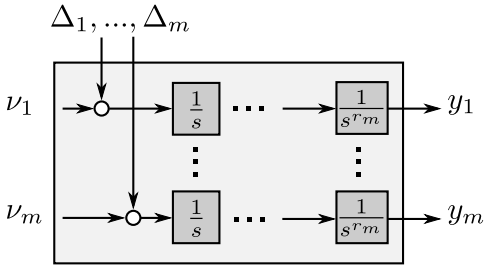


Abbildung 4.8.: Darstellung des vereinfachten Ersatzsystems mit Inversionsfehler.

Um die Fehlerdynamik weiter zu quantifizieren, wird der sogenannte Regelfehler $\vec{e}$ eingeführt:

$$\vec{e} = \begin{bmatrix} e_1 \\ \vdots \\ e_m \end{bmatrix} = \begin{bmatrix} y_1 - y_{ref,1} \\ \vdots \\ y_m - y_{ref,m} \end{bmatrix}. \tag{4.42}$$

Im Gegensatz zum Inversionsfehler $\vec{\Delta}$ wird dieser nicht aus den Modellsystemen abgeleitet, sondern direkt als Differenz aus dem Systemausgang $\vec{y}$ und dem Referenzsignal $\vec{y}_{ref}$ bestimmt. Die zeitlichen Ableitungen des Regelfehlers können sukzessive bis zum relativen Grad r_i durchgeführt werden:

$$[e_i{}^{r_i}]_{m\times 1} = [y_i^{(r_i)}]_{m\times 1} - [y_{ref,i}^{(r_i)}]_{m\times 1}. \tag{4.43}$$

Um die folgenden Veranschaulichungen zu vereinfachen, soll wieder das SISO-System betrachtet werden. Die Ergebnisse können leicht auf MIMO-Systeme übertragen werden. Wie bereits beschrieben, hat der Inversionsfehler Δ einen Regelfehler e zur Folge. Um dem Regelfehler entgegenzuwirken, wird eine Möglichkeit zur Rückführung dieses Fehlers vorgestellt. Dazu wird, ähnlich zur Gleichung (4.11) der Byrnes-Isidory-Normalform, zusätzlich zum Ausgangssignal das Referenzsignal mit seinen Ableitungen im transformierten Koordinatensystem dargestellt:

$$\vec{\xi}_{ref} = \begin{bmatrix} y_{ref} \\ \dot{y}_{ref} \\ \vdots \\ y_{ref}^{(r-1)} \end{bmatrix} \quad \text{und} \quad \vec{\xi} = \begin{bmatrix} y \\ \dot{y} \\ \vdots \\ y^{(r-1)} \end{bmatrix}. \tag{4.44}$$

Dabei entspricht die Differenz den transformierten Größen, den Ableitungen des Fehlers e bis zum relativen Grad $(r-1)$:

$$\vec{\xi} - \vec{\xi}_{ref} = \begin{bmatrix} e \\ \dot{e} \\ \vdots \\ e^{(r-1)} \end{bmatrix}. \tag{4.45}$$

Mit dem Ziel der Folgeregelung werden nun die Fehlerableitungen mit den Parametern c_i vom Referenzsignal $y_{ref}^{(r)}$ subtrahiert, um die neue Pseudosteuergröße ν zu bilden.

$$\begin{aligned}\nu &= y_{ref}^{(r)} - c_{r-1} \cdot e^{(r-1)} - \ldots - c_0 \cdot e^{(0)} \\ &= y_{ref}^{(r)} - \vec{c}^{\,T} \cdot [\vec{\xi} - \vec{\xi}_{ref}].\end{aligned} \tag{4.46}$$

In Abbildung 4.9 wird die neue Regelarchitektur dargestellt. Deutlich zu sehen ist die Rückführung der Zustandsgrößen $\vec{x}$ der E/A-Linearisierung sowie die Rückführung der Ausgangsableitungen $\vec{\xi}$ zum Fehlerregler. Wohlgemerkt wird hierbei noch nicht auf die Begrenzungen der Aktuatorik eingegangen, dies soll im nächsten Abschnitt erörtert werden. Dennoch bildet die vorgestellte Reglerarchitektur den Grundstein der weiteren Untersuchungen ab.

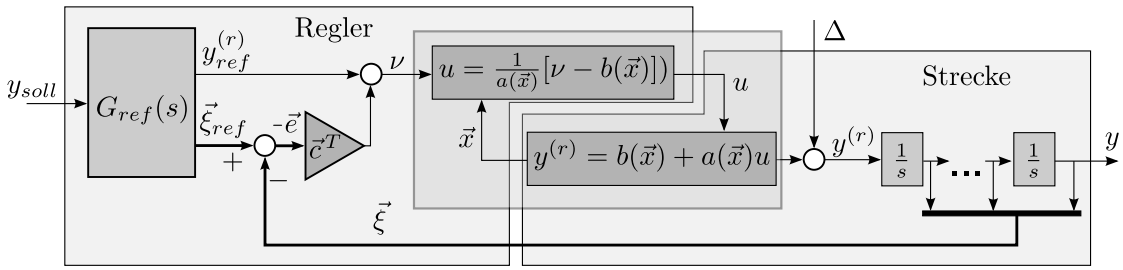


Abbildung 4.9.: Referenzmodell und Rückführung der Ausgangsgrößen im Fehlerregler (SISO).

Mit der erweiterten Pseudosteuergröße ν kann nun ein Weg zur Betrachtung der entstandenen Fehlerdynamik gefunden werden. Dazu wird, analog zu Gleichung (4.40) mit $i = 1$ (SISO), die neue Pseudosteuergröße (4.46) eingesetzt:

$$\begin{aligned}y^{(r)} &= \nu + \Delta \\ &= y_{ref}^{(r)} - c_{r-1} \cdot e^{(r-1)} - \ldots - c_0 \cdot e^{(0)} + \Delta.\end{aligned} \tag{4.47}$$

Im SISO-Fall gilt nach (4.43): $e^{(r)} = y^{(r)} - y_{ref}^{(r)}$, womit eine Differentialgleichung für die Fehlerdynamik aus Gleichung (4.47) entsteht:

$$e^{(r)} + c_{r-1} \cdot e^{(r-1)} + \ldots + c_0 \cdot e = \Delta. \tag{4.48}$$

Die Parameter c_i können als Reglerparameter für die Fehlerdynamik frei gewählt werden, um ein stabilisierendes Verhalten zu gewinnen. Das bedeutet, dass trotz Abweichungen zwischen System und fehlerhaftem Modell, der Regelfehler in Grenzen

gehalten wird. Da die Fehlerdynamik für die Stabilität der Reglerarchitektur von immenser Bedeutung ist, soll diese noch einen Schritt weiter zusammengefasst und betrachtet werden.

Mit dem Ziel einer Zustandsraumdarstellung wird der Vektor

$$\vec{\chi}(t) = [e(t), \dot{e}(t), ..., e^{(r-1)}(t)]^T \tag{4.49}$$

eingeführt. Daraus bildet sich mit Gleichung (4.48) die Differentialgleichung der Form

$$\begin{aligned} \dot{\vec{\chi}} &= \mathbf{A}_E \cdot \vec{\chi} + \mathbf{b}_E \cdot \Delta \\ \dot{\vec{\eta}} &= q(\vec{\xi}_{ref}(t) + \vec{\chi}, \vec{\eta}). \end{aligned} \tag{4.50}$$

Dabei entsprechen die Matrix $\mathbf{A}_E$ sowie der Vektor $\mathbf{b}_E$:

$$\mathbf{A}_E = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ -c_0 & -c_1 & -c_2 & \dots & -c_{r-1} \end{bmatrix}, \quad \mathbf{b}_E = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}. \tag{4.51}$$

An dieser Stelle sei angemerkt, dass auch die interne Dynamik vom Fehlervektor $\vec{\chi}$ beeinflusst wird.

Bereits in [Isi95] konnte mit der Voraussetzung von positiven Elementen des Parametervektors $\vec{c} = [c_{r-1}, ..., c_0]$ ein ausführlicher Stabilitätsnachweis für die Fehlerdynamik vorgestellt werden. Nach [Hol04, Krü12] konnte zudem gezeigt werden, dass für einen maximalen Inversionsfehler Δ eine maximale Schranke für den Fehlervektor $\vec{\chi}$ existiert. Dennoch kommt diese Fehlerrückführung an seine Grenzen, besonders im Fall von stark abweichenden Systemen, wie z.B. der Degeneration des Systems, also einer starken, unerwarteten Veränderung des E/A-Verhaltens. Um diesen Effekten entgegenzuwirken, sollen im späteren Verlauf dieser Arbeit die neuronalen Netze als adaptive Elemente vorgestellt werden. Dabei soll der Regelfehler e wieder betrachtet werden.

4.7 Pseudo-Control-Hedging

Mit der Fehlerrückführung wurde im letzten Kapitel mit dem Inversionsfehler Δ, einer Abweichung zwischen Modell und Strecke, Rechnung getragen. Damit ist es gelungen, das System trotz fehlerhafter Modellierung zu stabilisieren. Fehler können unter anderem durch Begrenzungen und Verzögerungen in der Aktuatorik verursacht werden. Zudem zeigt sich, dass gerade Aktuatorbegrenzungen zu destabilisierenden Effekten führen können [JC00]. Mit der Möglichkeit der Messung des Stellsignals oder einer hinreichenden Modellierung der Aktuatorik, können die durch die Aktuatorik verursachten Fehler in der Fehlerdynamik berücksichtigt werden. Dazu wurde von [JC00] das Konzept des Pseudo-Control-Hedging vorgestellt.

Die Grundidee dieses Konzeptes kann, anhand dem aus der Regelungstechnik bekannten „Anti-Windup“, illustriert werden (siehe Grafik 4.10). Dabei wird die Differenz zwischen Eingang und Ausgang des Aktuators, hier als einfache Begrenzung $G_A(t)$ dargestellt, zurückgeführt. Dies hat den Vorteil, dass der durch die Begrenzung entstehende Fehler nicht weiter im Integrator des Reglers berücksichtigt wird. Ohne Anti-Windup könnte der Integrator Stellsignale u_k generieren, denen der Ak-

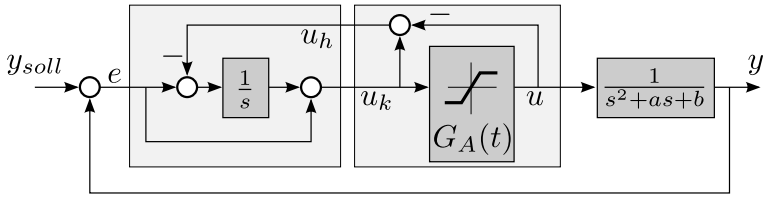


Abbildung 4.10.: Einfache Anti-Windup Konfiguration.

tuator nicht mehr folgen kann, was zu instabilem Verhalten führt. Einem ähnlichen Prinzip bedient sich das sogenannte Pseudo-Control-Hedging (PCH).

Um eine bessere Darstellung zu ermöglichen, soll angelehnt an [Hol04], eine neue Notation eingeführt werden. Dazu ersetzen die Substitutionen

$$\begin{aligned} \mathbf{F}(\vec{x}, u) &= b(\vec{x}) + a(\vec{x}) \cdot u \\ \hat{\mathbf{F}}(\vec{x}, u) &= \hat{b}(\vec{x}) + \hat{a}(\vec{x}) \cdot u \\ \hat{\mathbf{F}}^{-1}(\vec{x}, \vec{\nu}) &= \hat{a}^{-1}(\vec{x})[\vec{\nu} - \hat{b}(\vec{x})] \end{aligned} \tag{4.52}$$

die jeweiligen Gleichungen der Byrnes-Isidory-Normalform sowie der entsprechenden Inversen (Gleichung (4.21) und (4.27)). Anhand von Abbildung 4.11 soll die Funktionsweise des PCH für den SISO-Fall dargestellt werden. Angemerkt sei an dieser

Stelle, dass die bereits beschriebene Fehlerrückführung und das Referenzmodell in $G_\nu(s)$ zusammengefasst wurde. Der Ausgang $y^{(r)}$ des Systems setzt sich aus der

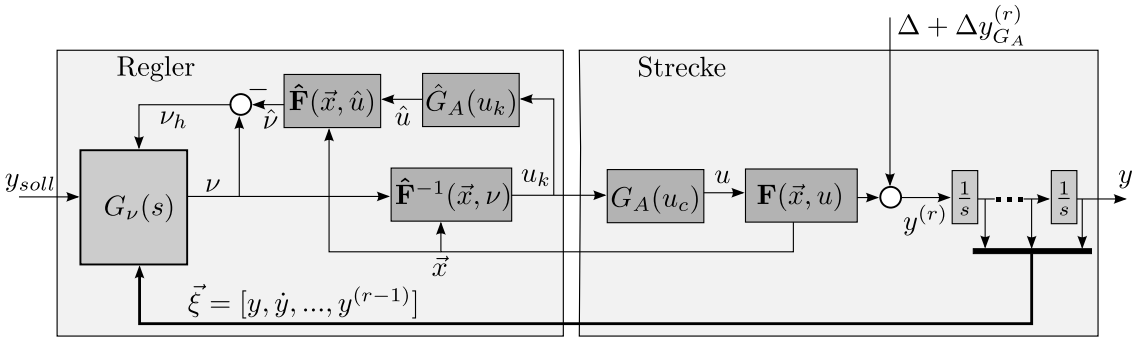


Abbildung 4.11.: Zusammenfassung der Inversion mit Pseudo-Control-Hedging (PCH), Fehlerregler und Referenzmodell.

Pseudosteuergröße ν und dem Inversionsfehler Δ nach Gleichung (4.47) zusammen. Da bis dato Abweichungen bzgl. der Aktuatorik nicht betrachtet wurden, sollen diese nun mit der Einführung von $\Delta y_{G_A}^{(r)}$ vom Inversionsfehler separiert werden. So ergibt sich nun für den Ausgang $y^{(r)}$:

$$y^{(r)} = \nu + \Delta + \Delta y_{G_A}^{(r)}. \tag{4.53}$$

Man kann somit sagen, dass der Fehler $\Delta y_{G_A}^{(r)}$, der Abweichung zwischen dem Ausgang mit bzw. ohne berücksichtigter Aktuatorik entspricht:

$$y^{(r)}(\vec{x}, G_A(u_k)) = y^{(r)}(\vec{x}, u_k) + \Delta y_{G_A}^{(r)}. \tag{4.54}$$

Um nun, ähnlich dem Anti-Windup, eine Rückführung der Differenz zwischen Stellsignal und Stellgröße zu ermöglichen, muss diese in die Pseudosteuergröße umgerechnet werden. Kann die Stellgröße u gemessen oder mit $\hat{u} = u$ annähernd genau berechnet werden, so wird mit dem Modell $\hat{\mathbf{F}}$,

$$\hat{\nu} = \hat{\mathbf{F}}(\vec{x}, \hat{u}) \tag{4.55}$$

angenommen, eine Pseudosteuergröße $\hat{\nu}$ erstellen zu können. Damit entspricht diese berechnete Pseudosteuergröße der Steuergröße, der das System inkl. Aktuatorik tatsächlich folgen konnte, d.h. der Aktuatorfehler ist hier nicht mehr vorhanden:

$$y^{(r)} = \hat{\nu} + \Delta. \tag{4.56}$$

Mit dem sogenannten Hedge-Signal ν_h kann die Abweichung $\Delta y_{G_A}^{(r)}$ jetzt rekonstruiert werden:

$$\nu_h = \nu - \hat{\nu} = -\Delta y_{G_A}^{(r)}. \tag{4.57}$$

Damit ermöglicht die Berücksichtigung der Aktuatordynamik eine neue Betrachtung der Fehlerdynamik (4.48). Betrachtet man dazu noch einmal die Pseudosteuergröße (4.46) in Abhängigkeit vom Referenzsignal y_{ref} sowie vom Fehlervektor $\vec{\chi}$:

$$\begin{aligned} \nu &= y_{ref}^{(r)} - c_{r-1} \cdot e^{(r-1)} - \ldots - c_0 \cdot e^{(0)} \\ &= y_{ref}^{(r)} - \vec{c}^{\,T} \cdot \vec{\chi}, \end{aligned} \tag{4.58}$$

so kann die Abweichung des gewünschten Referenzsignals in $\hat{\nu}$ wie folgt dargestellt werden:

$$\hat{\nu} = y_{ref}^{(r)} - \vec{c}^{\,T} \cdot \vec{\chi} + \Delta y_{ref,G_A}^{(r)} - \vec{c}^{\,T} \cdot \Delta\vec{\chi}_{G_A}. \tag{4.59}$$

Das bedeutet, dass mit dem Hedge-Signal

$$\nu_h = \vec{c}^{\,T} \cdot \Delta\vec{\chi}_{G_A} - \Delta y_{ref,G_A}^{(r)}, \tag{4.60}$$

der nicht erreichbare Anteil der kommandierten Pseudosteuergröße ν herausgerechnet wurde.

Um die neue Betrachtung der Fehlerdynamik zu vereinfachen, wird das Schaubild 4.12 des, um PCH erweiterten Referenzmodells mit Fehlerrückführung eingeführt.

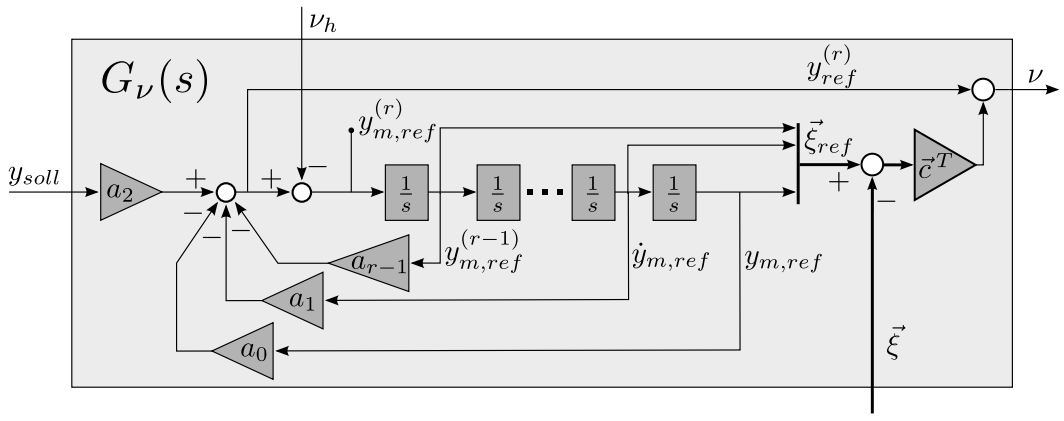


Abbildung 4.12.: Referenzmodell mit Fehlerrückführung und PCH.

Da der Ausgang nach Gleichung (4.53), bedingt durch eine Verzögerung oder Beschränkung in der Aktuatorik, der Pseudosteuergröße ν nicht folgen kann, wird mit

$$y_{m,ref}^{(r)} = y_{ref}^{(r)} - \nu_h \tag{4.61}$$

ein Messpunkt eingeführt.

Mit der auf die Pseudosteuergröße zurückgeführten Aktuatorabweichung (4.57), eingesetzt in die Ausgangsgleichung (4.53), folgt:

$$y^{(r)} = \nu - \nu_h + \Delta. \tag{4.62}$$

Hierfür kann man sich auch bewusst machen, dass die Inversion mit

$$y^{(r)} = \hat{\mathbf{F}}^{-1}(\vec{x}, u)\mathbf{F}(\vec{x}, \nu) \neq \nu + \Delta \tag{4.63}$$

im Fall des Aktuatorfehlers bezüglich der reinen Pseudosteuergröße ν ungültig geworden ist. Kombiniert man die Gleichung (4.62) mit der Gleichung (4.58) für die Pseudosteuergröße

$$y^{(r)} = y_{ref}^{(r)} - \vec{c}^{\,T} \cdot \vec{\chi} - \nu_h + \Delta, \tag{4.64}$$

kann die Gleichung gegenüber dem Messpunkt (4.61) aufgelöst werden:

$$y^{(r)} = y_{m,ref}^{(r)} - \vec{c}^{\,T} \cdot \vec{\chi} + \Delta. \tag{4.65}$$

Dies ist eine abweichende Notation zu [Hol04], um Verwechslungen mit der zuvor verwendeten Notation aus Gleichung (4.40) zu vermeiden.

Diese Betrachtungsweise erlaubt folgende Interpretation: Statt dem Referenzsignal $y_{ref}^{(r)}$ zu folgen, dem die Strecke wegen der Aktuatordynamik und Aktuatorbegrenzungen nicht folgen kann, folgt der Referenzausgang $y^{(r)}$ dem Messpunkt $y_{m,ref}^{(r)}$. Betrachtet man

$$y^{(r)} - y_{m,ref}^{(r)} = e^{(r)} \tag{4.66}$$

als neuen Regelfehler (vgl. Gleichung (4.43)), so hat die Aktuatorabweichung keinen Einfluss mehr auf diesen Fehler. Da auch die Ableitungen $\vec{\chi}$ des Fehlers aus diesem Fehler gebildet werden, konnte mit dem PCH eine Fehlerdynamik ohne Einfluss

des Aktuators gefunden werden. Offensichtlich gilt die Gleichung der Fehlerdynamik (4.48) sowie die Differenzialgleichung (4.50) bezüglich des Messpunktes $y_{m,ref}^{(r)}$ weiterhin:

$$\begin{aligned} \Delta &= e^{(r)} + \vec{c}^{\,T} \cdot \vec{\chi} \\ \dot{\vec{\chi}} &= \mathbf{A}_E \cdot \vec{\chi} + \mathbf{b}_E \cdot \Delta \\ \dot{\vec{\eta}} &= q(\vec{\xi}_{ref}(t) + \vec{\chi}, \vec{\eta}). \end{aligned} \tag{4.67}$$

Gerade in Szenarien, in denen das System an seine Grenzen kommt, z.B. im Degradationsfall, ist das PCH von immenser Bedeutung. Da die Aktuatorbegrenzungen im Fehler e nicht sichtbar sind, wird verhindert, dass die neuronalen Netze nicht die Begrenzungen der Aktuatoren erlernen. Das bedeutet, dass diese adaptiven Elemente nur den Inversionsfehler akkumulieren [JC00, Hol04].

Mit dem PCH wurde weiterhin eine Möglichkeit gefunden, destabilisierende Effekte durch Aktuatorbegrenzungen bzw. Verzögerungen zu verhindern. Das bedeutet, ähnlich wie beim Anti-Windup, dass die Zustände des Referenzmodells durch Stellbegrenzungen nicht unbegrenzt ansteigen.

An dieser Stelle muss erwähnt werden, dass mit dem PCH auch Nachteile „erkauft“ werden, wenn die Stellgröße nicht direkt gemessen werden kann. Im Fall eines Aktuatorfehlers, d.h. wenn der Aktuator ein vom vorhergesehenen Verhalten abweichendes Verhalten annimmt, werden die Fehler direkt in die übergeordnete Kaskade übertragen, womit das Gesamtübertragungsverhalten beeinflusst wird.

4.8 Umsetzung der dynamischen Inversion

Nachdem in den vorangegangenen Kapiteln die theoretischen Grundlagen der dynamischen Inversion und des Pseudo-Control-Hedgings dargestellt wurden, soll nun die Anwendung auf das konkrete E/A-Verhalten der Flugzeugdynamik aus Kapitel 3 erfolgen. Grundsätzlich gibt es mehrere Ansätze zur Umsetzung der dynamischen Inversion. Die folgenden Zusammenhänge werden einleitend mit der vereinfachten Skizze 4.13, des in Kapitel 3 beschriebenen E/A-Verhaltens des unbemannten Flugzeuges T200, dargestellt. Die abgebildete Flugzeugdynamik beinhaltet die Rotationsdynamik $\vec{\omega}_? = \mathbf{F}_{\vec{\omega}}(\vec{\omega}_?, \vec{u})$ sowie die Lagedynamik $\vec{\Omega}_? = \mathbf{F}_{\vec{\Omega}}(\vec{\Omega}_?, \vec{\omega}_?)$. Die Gesamtdynamik $\mathbf{F}(\vec{x}, \vec{u})$ kann als Hintereinanderschaltung der beiden Dynamiken mit

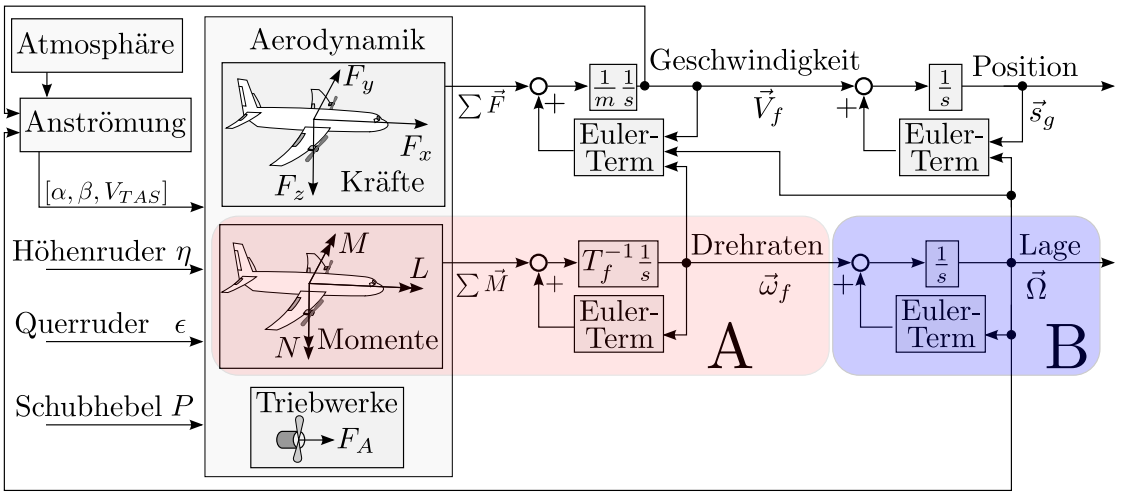


Abbildung 4.13.: Vereinfachte Darstellung der Flugzeugsimulation.

$$\dot{\vec{x}} = \mathbf{F}(\vec{x}, \vec{u}) = \mathbf{F}_{\vec{\Omega}}(F_{\vec{\omega}}(\vec{\omega}_?, \vec{u}), \vec{\Omega}_?) \tag{4.68}$$

dargestellt werden, wobei $\vec{x}$ und $\vec{u}$ einem zu definierenden Zustands- und Eingangsvektor entspricht. Die Arten der dynamischen Inversion unterscheiden sich in der Wahl des Koordinatensystems, wie hier mit dem undefinierten Indize „?“ angedeutet wird und in der Wahl des relativen Grades.

Allen Varianten der Inversionsregelung ist die Invertierung der Lage- und Rotationsdynamik gemein, die z.B. als innere Kaskade eines Bahnreglers arbeitet. Betrachtet man noch einmal Gleichung 4.68, so wird deutlich, dass die Realisierung des Reglers nun mit dem relativen Grad eins durch separate Inversion der Rotationsdynamik und der Lagedynamik erfolgen kann oder durch die Inversion der Gesamtdynamik, was als Inversion mit relativen Grad zwei bezeichnet wird. Bei der Inversion mit dem Grad zwei zeigen die vorhandenen Arbeiten [Hol04, Krü12], dass die Fehlerdynamik von entsprechend hohen Ableitungen der Fehlergrößen abhängig ist, diese aber nicht direkt gemessen werden können und so auf spezielle Filterverfahren zurückgegriffen werden muss. Zusätzlich wird die mathematische Umsetzung der Inversion zweiten Grades deutlich aufwendiger [Krü12]. Es zeigt sich, dass die mit der Methode zweiten Grades gewonnene höhere Übertragungsbandbreite mit Kopplungseffekten einhergeht, die zu Bahnabweichungen führen kann [Rob13]. Gegenüber diesen Nachteilen konnten positive Effekte der Inversion ersten Grades in Kombination mit dem, im nächsten Abschnitt vorgestellten, Gleitzustandslernverfahren dargestellt werden [Krü12]. Dabei zeigt sich, dass die Zustände der Fehlerdynamik, sowie deren Ableitungen eine höhere Robustheit gegenüber äußeren Störungen aufweisen.

Neben dem relativen Grad existiert für die dynamische Inversion die Variation des Koordinatensystems. Je nach Betrachtungsweise können die zu regelnden Systemzustände mit den Eulerwinkeln und Drehraten im körperfesten Koordinatensystem

$$\vec{x} = [\Phi, \Theta, \Psi, p, q, r]^T \tag{4.69}$$

oder den Bahnwinkeln und Drehraten im aerodynamischen Koordinatensystem

$$\vec{x} = [\alpha_K, \beta_K, \mu_K, p_A, q_A, r_A]^T, \tag{4.70}$$

als Zustände angenommen werden [Now10, Krü12]. Beim unbemannten Luftfahrzeug T200 gelten zudem Höhenruder, Querruder und der Schub als Eingang $\vec{u} = [\eta, \epsilon, P]^T$ des Systems. Ein Seitenruder steht nicht zur Verfügung. Die Regelung der Bahnwinkel hat den Vorteil, dass der Anstellwinkel α sowie der Schiebewinkel β berücksichtigt werden. Die Inversion der Bahnwinkel wurde in [Krü12, Sch11b, Now10] ausführlich in der Simulation und im Windkanal-Freiflug getestet. Nachteil der Bahnwinkelinversion ist die Voraussetzung einer Messeinrichtung für Anstell- sowie Schiebewinkel. Das bedeutet auf der anderen Seite, dass Anstellwinkel und Schiebewinkel im Fall einer Inversion der Eulerwinkel in die interne Dynamik fallen und diese, wie bereits erwähnt, eine Eigenstabilität besitzen müssen. Begünstigend für die Inversion der Eulerwinkel ist die Tatsache, dass diese mit hoher Abtastrate direkt aus der inertialen Messeinheit gewonnen werden. Zudem wurde die Eulerwinkelinversion bereits erfolgreich im Flugversuch erprobt [JL01, Loo08]. Zwar ist die Messung von Anstellwinkel und Schiebewinkel mit einer sogenannten 5-Loch Sonde in unbemannten Flugzeugen prinzipiell realisierbar [KB12], eine Einbindung in die Autopiloten-Architektur war aber bis zum gegebenen Zeitpunkt nicht möglich. Zudem kann der Autopilot wegen seinem geringen Gewicht ohne 5-Loch Sonde auch in handelsüblichen Elapor (Schaumstoff) Modellflugzeugen eingebaut werden, einer Anordnung, die die experimentelle Erstinbetriebnahme besonders begünstigt. Die folgenden Untersuchungen konzentrieren sich aus den dargestellten Gründen auf eine Inversion der Eulerwinkel mit dem relativen Grad eins. Im Folgenden werden weiterhin die flugzeugfesten Koordinaten für die Drehraten und die Lagewinkel verwendet. Aus Gründen der Übersichtlichkeit wird bei diesen Größen auf die Darstellung der Indize verzichtet. Damit gilt die Definition der messbaren Systemzustände:

$$\vec{x} = \begin{bmatrix} \vec{\Omega}_f \\ \vec{\omega}_f^{gf} \end{bmatrix} = \begin{bmatrix} \vec{\Omega} \\ \vec{\omega} \end{bmatrix} = [\Phi, \Theta, \Psi, p, q, r]^T. \tag{4.71}$$

Einführend skizziert Abbildung 4.14 alle Bestandteile der verwendeten Inversionsform. Dabei ist zu beachten, dass die Systemzustände $\vec{x}$ auf der Reglerseite, der fehlerbehafteten Messung entsprechen (vgl. Kapitel 2.2 und 3.3). Dies wird in der Simulation berücksichtigt. Angelehnt an [Krü12, Hol04] wird auf eine Unterscheidung zwischen Messung und realer Größe, anhand von Indize, verzichtet. Die weitere Darstellung des „Messung“-Blocks (vgl. Abbildung 4.14), wird zu Gunsten der Übersichtlichkeit vernachlässigt.

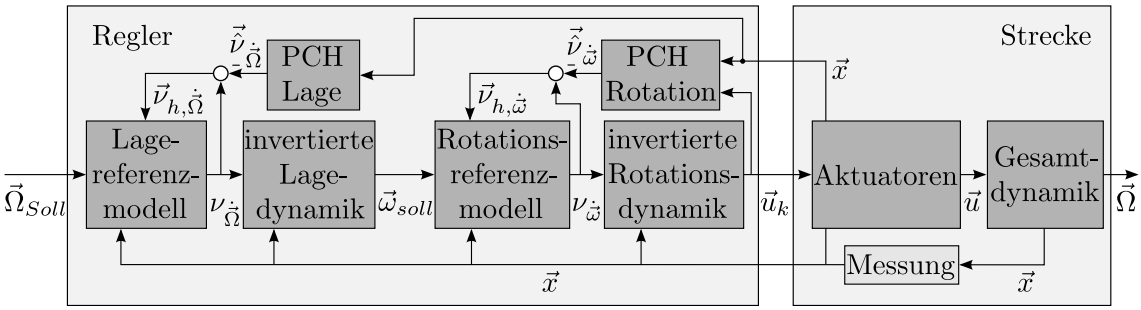


Abbildung 4.14.: Übersicht der Eulerinversion mit relativem Grad eins.

4.8.1 Rotationsdynamik

Mit der Wahl der Reglerstruktur im letzten Abschnitt folgt die Definition der Inversion der Rotationsdynamik. Diese wird zuerst allgemein formuliert und dann für die, mit der Flugzeugdefinition entstehenden Einschränkungen, optimiert. Als Ausgangsgleichung dieser Inversion soll die Rotationsdynamik aus Gleichung (3.10) noch einmal betrachtet werden:

$$\dot{\vec{\omega}} = \mathbf{T}_f^{-1} \cdot \left[\sum \vec{M}_f - \vec{\omega} \times (\mathbf{T}_f \cdot \vec{\omega}) \right]. \tag{4.72}$$

Betrachtet man $\dot{\vec{\omega}}$ als Ausgang, so gilt für Gleichung (4.72) nach dem in Kapitel 4.2 beschriebenen Vorgehen, offensichtlich der relative Grad $r_{\vec{\omega}} = 1$ für die Rotationsdynamik. Durch Umstellen nach den Momenten $\sum \vec{M}_f$ wird die Inversion dieser Dynamik erreicht:

$$\sum \vec{M}_f = \mathbf{T}_f \cdot \vec{\nu}_{\dot{\vec{\omega}}} + \vec{\omega} \times (\mathbf{T}_f \cdot \vec{\omega}), \tag{4.73}$$

wobei $\vec{\nu}_{\dot{\vec{\omega}}}$ einer Pseudosteuergröße aus dem noch zu definierenden Referenzmodell entspricht. Die Differentialgleichung des Referenzmodells ergibt sich am Messpunkt

(vgl. Gleichung (4.61)) mit der gewünschten Drehrate $\vec{\omega}_{soll}$ und dem Hedge-Signal $\vec{\nu}_{h,\dot{\vec{\omega}}} = \vec{\nu}_{\dot{\vec{\omega}}} - \hat{\vec{\nu}}_{\dot{\vec{\omega}}}$, nach dem Schema des Referenzmodells mit PCH, zu

$$\dot{\vec{\omega}}_{m,ref} = \tilde{\mathbf{T}}_{\vec{\omega}} \cdot (\vec{\omega}_{soll} - \vec{\omega}_{m,ref}) - \vec{\nu}_{h,\dot{\vec{\omega}}}, \tag{4.74}$$

wobei die Zeitkonstanten der gewünschten Rotationsdynamiken mit der Matrix

$$\tilde{\mathbf{T}}_{\vec{\omega}} = \begin{bmatrix} \frac{1}{T_p} & 0 & 0 \\ 0 & \frac{1}{T_q} & 0 \\ 0 & 0 & \frac{1}{T_r} \end{bmatrix} \tag{4.75}$$

vorgegeben werden. Diese Zeitkonstante repräsentiert die Geschwindigkeit des Referenzmodells und damit die Geschwindigkeit der Reaktion auf Sollwertvorgaben. Sie beeinflusst aber nicht die Dynamik des Fehlerreglers, da das Referenzmodell keine Rückführung des Fehlers beinhaltet.

Die Pseudosteuergröße resultiert aus der Referenzgröße und dem Fehlerregler:

$$\vec{\nu}_{\dot{\vec{\omega}}} = \tilde{\mathbf{T}}_{\vec{\omega}} \cdot (\vec{\omega}_{soll} - \vec{\omega}_{m,ref}) + \mathbf{K}_{\vec{\omega}} \cdot (\vec{\omega}_{m,ref} - \vec{\omega}), \tag{4.76}$$

mit der Matrix $K_{\vec{\omega}}$ der Verstärkungsfaktoren für die Fehlerrückführung, die folgendermaßen definiert ist:

$$\mathbf{K}_{\vec{\omega}} = \begin{bmatrix} K_p & 0 & 0 \\ 0 & K_q & 0 \\ 0 & 0 & K_r \end{bmatrix}. \tag{4.77}$$

Mit diesen Zeitkonstanten wird die Geschwindigkeit der Reaktion auf Fehlergrößen vorgegeben. Damit kann, in Anlehnung an den kaskadischen Aufbau linearer Flugregelungssysteme, die hier entstehende Drehratenregelung als innerste Schleife und damit als Dämpfer interpretiert werden [Krü12, BAL11]. Wie beim linearen Regler ist bei der Auslegung der Zeitkonstantenmatrizen $\tilde{\mathbf{T}}_{\omega}$ und $\mathbf{K}_{\vec{\omega}}$ auf genügend Abstand zwischen den einzelnen Kaskaden zu achten. Daraus folgt, dass die innere Kaskade, also die entstandene Rotationsdynamik, schneller sein muss, als die außen liegende Lagedynamik. Theoretisch, ohne physikalische Begrenzungen in Aktuatorik und Material, kann hier jede beliebig schnelle Dynamik vorgegeben werden. Im Realfall können diese Begrenzungen nicht vernachlässigt werden und bilden die limitierenden Faktoren.

Die Abbildung 4.15 stellt zusammenfassend die Struktur des Referenzmodells $G_{\vec{\nu},\vec{\omega}}(s)$ mit der Fehlerrückführung und dem Hedge-Signal dar.

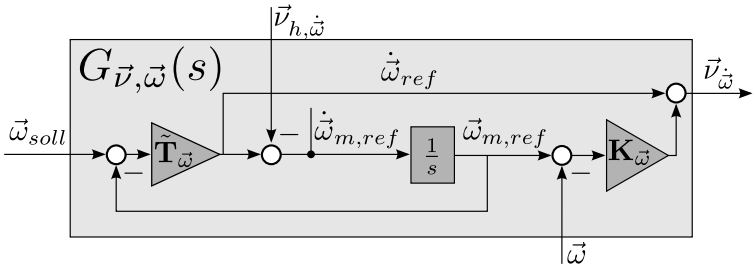


Abbildung 4.15.: Referenzmodell erster Ordnung mit Fehlerrückführung.

Wie bereits beschrieben, wird das Flugzeugmodell und in Folge dessen die Inversion vom realen Flugzeug abweichen. Dieser Fehler wird im Fehlerregler minimiert, kann aber, wie später eingeführt, mit adaptiven Elementen erlernt und kompensiert werden. Um einen Zusammenhang zu den im nächsten Unterkapitel vorgestellten adaptiven Elementen in Form von künstlichen neuronalen Netzen herzustellen, wird die Fehlerdynamik bestimmt. Mit dem Fehler

$$\vec{e}_{\vec{\omega}} = \begin{bmatrix} e_p \\ e_q \\ e_r \end{bmatrix} = \vec{\omega} - \vec{\omega}_{m,ref} = \begin{bmatrix} p - p_{m,ref} \\ q - q_{m,ref} \\ r - r_{m,ref} \end{bmatrix}, \tag{4.78}$$

als Differenz zwischen Messgröße und dem Referenzsignal im Messpunkt, wird analog zum Theorieabschnitt, vgl. Gleichung (4.67), die Gleichung der Fehlerdynamik gebildet:

$$\dot{\vec{e}}_{\vec{\omega}} + \mathbf{K}_{\vec{\omega}} \cdot \vec{e}_{\vec{\omega}} = \vec{\Delta}_{\vec{\omega}}. \tag{4.79}$$

Mit der Grafik 4.16 wird der gesamte Aufbau der Rotationsregelung dargestellt. An dieser Stelle sei angemerkt, dass angelehnt an [Hol04, Krü12, Rob13] ein inkrementeller Ansatz mit Steuerflächenzuweisung verwendet wird. Dabei wird das auf das Flugzeug wirkende Gesamtmoment aufgeteilt, in das Moment des aktuellen Flugzustandes und in ein Momenteninkrement, das erforderlich ist, um den gewünschten Flugzustand zu erreichen:

$$\sum \vec{M}_f = \underbrace{\left[\vec{M}_0(\vec{\omega}, \vec{u}_k)\right]_f}_{\text{aktueller Flugzustand}} + \underbrace{\left[\delta\vec{M}_k(\vec{\omega}, \vec{\nu}_{\dot{\vec{\omega}}})\right]_f}_{\text{erforderliches Inkrement}}. \tag{4.80}$$

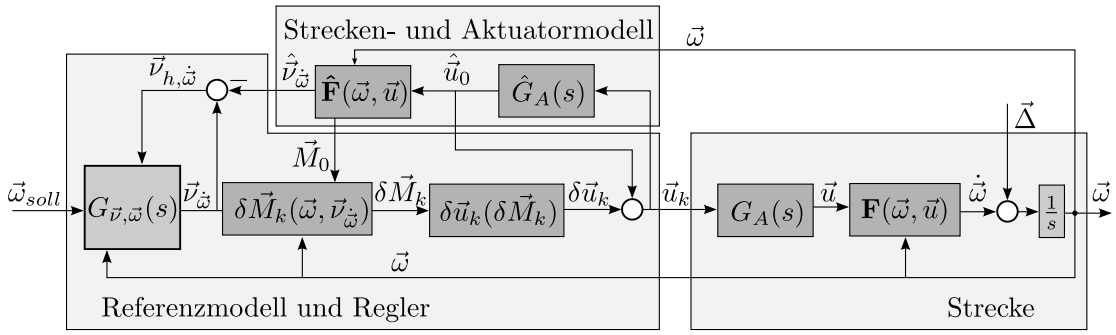


Abbildung 4.16.: Übersicht der inneren Kaskade der Eulerinversion mit relativem Grad eins.

Das erforderliche Inkrement

$$\left[\delta\vec{M}_k(\vec{\omega},\vec{\nu}_{\dot{\vec{\omega}}})\right]_f = \begin{bmatrix}\delta L_k\\ \delta M_k\\ \delta N_k\end{bmatrix}, \tag{4.81}$$

entsteht durch Einsetzen von Gleichung (4.80) in den invertierten Drehimpulssatz (4.73):

$$\left[\delta\vec{M}_k(\vec{\omega},\vec{\nu}_{\dot{\vec{\omega}}})\right]_f = \mathbf{T}_f\cdot\vec{\nu}_{\dot{\vec{\omega}}} + \vec{\omega}\times(\mathbf{T}_f\cdot\vec{\omega}) - \left[\vec{M}_0(\vec{x},\vec{u})\right]_f. \tag{4.82}$$

Der Berechnung des aktuellen Momentenhaushaltes $\left[\vec{M}_0(\vec{x},\vec{u})\right]_f$ des Flugzustandes wird mit dem bereits im PCH simulierten Modell $\hat{\mathbf{F}}(\vec{\omega},\vec{u})$ sowie der Aktuatorik $\hat{G}_A(s)$ ermöglicht. Die Werte ergeben sich also direkt aus dem parallel zum Flugzeug simulierten System, vgl. Gleichung (3.20). Diese Methode erweist sich als besonders effizient, da hierfür keine zusätzlichen Simulationen benötigt werden. Als vereinfachende Annahme wird der Anstellwinkel α und der Schiebewinkel β zu Null gesetzt.

Um letztendlich die Stellgröße u_k zu erhalten, wird die sogenannte Steuerflächenzuweisung bzw. Steuerflächenallokation verwendet. Damit wird der erforderlichen Momentenänderung $\delta\vec{M}_k$, die entsprechend benötigte Stellgröße $\vec{u}_k$ zugewiesen. Die kommandierte Stellgröße bzw. der kommandierte Stellausschlag ergibt sich direkt aus dem aktuellen geschätzten Stellausschlag $\hat{\vec{u}}_0$ und dem noch zu bestimmenden Inkrement $\delta\vec{u}_k$:

$$\vec{u}_k = \delta\vec{u}_k + \hat{\vec{u}}_0,\ \text{ mit }\ \delta\vec{u}_k = \begin{bmatrix}\delta\xi_k\\ \delta\eta_k\end{bmatrix}\ \text{ und }\ \hat{\vec{u}}_0 = \begin{bmatrix}\hat{\xi}_0\\ \hat{\eta}_0\end{bmatrix}. \tag{4.83}$$

Wie bereits erwähnt, stehen nur das Querruder ξ und das Höhenruder η zur Verfügung. Eine Steuerung der Gierrate ist zwar mit dem Querruder möglich, soll aber auf Grund von Kopplungseffekten mit dem Rollverhalten nicht in Betracht gezogen werden. Dennoch muss für die Momentenberechnung der Nick- und Rollratenregelung die Ersatzregelgröße $\nu_{\dot{r}}$ gebildet werden, da alle Drehraten und Momente über den Trägheitstensor miteinander verknüpft sind, wie in Gleichung (4.72) ersichtlich. Ein Lösungsansatz wird am Ende dieses Abschnitts dargestellt.

Um die Bestimmung der noch erforderlichen Stellgrößeninkremente zu ermöglichen, wird ein Zusammenhang zwischen den bereits berechneten Momenteninkrementen und den nötigen Beiwertinkrementen hergestellt:

$$\begin{aligned} \delta C_L &= \frac{\delta L_k}{\bar{q} \cdot S \cdot s} \\ \delta C_M &= \frac{\delta M_k}{\bar{q} \cdot S \cdot l_\mu}. \end{aligned} \tag{4.84}$$

Damit ergeben sich die Stellgrößeninkremente $\delta\vec{u}_k$ als Quotient der berechneten Momente sowie den Steuerflächenbeiwerten:

$$\begin{aligned} \delta\xi_k &= \frac{\delta C_L}{C_{L,\xi}} \\ \delta\eta_k &= \frac{\delta C_M}{C_{M,\eta}}. \end{aligned} \tag{4.85}$$

Um die zu erreichende Steuergröße für das PCH zu erhalten, kann die Drallgleichung mit dem zum nächsten Zeitpunk wirkenden Moment, unter Einfluss der Aktuatordynamik $G_A(s)$, erstellt werden:

$$\vec{\hat{\nu}}_{\dot{\vec{\omega}}} = \mathbf{T}_f^{-1} \cdot \left[\sum \vec{M}_{0,(t+1)}(G_A(s)) - \vec{\omega} \times (\mathbf{T}_f \cdot \vec{\omega}) \right]. \tag{4.86}$$

Das Hedge-Signal $\vec{\nu}_{h,\dot{\vec{\omega}}}$ bildet sich aus der Differenz der Pseudosteuergröße $\vec{\nu}_{\dot{\vec{\omega}}}$ und der geschätzten, zu erreichenden Pseudosteuergröße $\vec{\hat{\nu}}_{\dot{\vec{\omega}}}$,

$$\vec{\nu}_{h,\dot{\vec{\omega}}} = \vec{\nu}_{\dot{\vec{\omega}}} - \vec{\hat{\nu}}_{\dot{\vec{\omega}}}. \tag{4.87}$$

Die Problematik der nicht steuerbaren Gierratendynamik und der damit verbundenen, nicht aus Seitenrudersignalen berechenbaren Pseudosteuergröße $\nu_{\dot{r}}$ kann jetzt umgangen werden. Durch Verwendung der erreichten Steuergröße als Pseudosteuergröße $\nu_{\dot{r}} = \hat{\nu}_{\dot{r}}$ ist eine Momentenberechnung nach Gleichung 4.82 weiterhin möglich.

Ein Referenzmodell und ein Fehlerregler in der Gierrate sind im Fall eines fehlenden Seitenruders damit überflüssig.

4.8.2 Lagedynamik

Mit dem Rotationsregler wurde bereits die innerste Kaskade des zu implementierenden Bahnreglers erreicht. Die nächste äußere Schleife wird mit der folgend vorgestellten Lageregelung realisiert. Gleich dem Vorgehen wie bei der Rotationsdynamik, wird zunächst die Gleichung (3.13) zur Berechnung der Eulerwinkel noch einmal betrachtet:

$$\dot{\vec{\Omega}} = \mathbf{M}_{\Phi f} \cdot \vec{\omega}. \tag{4.88}$$

Die Inversion erfolgt durch das Umstellen der Gleichung (4.88) nach den Drehraten:

$$\vec{\omega}_{soll} = \mathbf{M}_{\Phi f}^{-1} \cdot \nu_{\dot{\vec{\Omega}}}. \tag{4.89}$$

In der ausmultiplizierten Darstellung ergibt sich die Form in Gleichung (4.90) und bildet damit den Eingang in die Rotationsregelung:

$$\begin{aligned} \vec{\omega}_{soll} = \begin{bmatrix} p_{soll} \\ q_{soll} \\ r_{soll} \end{bmatrix} &= \begin{bmatrix} 1 & 0 & -sin\Theta \\ 0 & cos\Phi & sin\Phi \cdot cos\Theta \\ 0 & -sin\Theta & cos\Phi \cdot \cos\Theta \end{bmatrix} \cdot \begin{bmatrix} \nu_{\dot{\Phi}} \\ \nu_{\dot{\Theta}} \\ \nu_{\dot{\Psi}} \end{bmatrix} \\ &= \begin{bmatrix} \nu_{\dot{\Phi}} - \nu_{\dot{\Psi}} \cdot sin\Theta \\ \nu_{\dot{\Theta}} \cdot cos\Phi + \nu_{\dot{\Psi}} \cdot sin\Phi \cdot cos\Theta \\ -\nu_{\dot{\Theta}} \cdot sin\Phi + \nu_{\dot{\Psi}} \cdot cos\Phi \cdot cos\Theta \end{bmatrix}. \end{aligned} \tag{4.90}$$

Die Soll-Gierrate ist im Fall eines fehlenden Seitenruders zwar überflüssig, wird aber zur besseren Darstellung beibehalten.

Eine Vereinfachung bietet sich an dieser Stelle für die Pseudosteuergröße $\nu_{\dot{\Psi}}$ des Gierwinkels an. Da keine Regelung des Gierwinkels erfolgt, kann analog zur Rotationsdynamik auf das Referenzmodell und den Fehlerregler der Gierdynamik verzichtet werden und stattdessen die Pseudosteuergröße direkt aus dem PCH entnommen werden:

$$\nu_{\dot{\Psi}} = \hat{\nu}_{\dot{\Psi}}. \tag{4.91}$$

Dies bedeutet, dass die Steuergröße für die Gierdynamik dem tatsächlich erfüllten $\dot{\Psi}$ entsprechen. Auch wenn keine Regelung der Gierdynamik erfolgt, ist diese Pseudosteuergröße für die Inversion in Gleichung (4.89) zur Berechnung der Roll- und Nickwinkel-Pseudosteuergrößen nötig.

Die Differenzialgleichung des Referenzmodells für die Eulerwinkel am Messpunkt m (vgl. Gleichung (4.61)), abzüglich des Hedge-Signals $\nu_{h,\dot{\vec{\Omega}}}$, wird analog zur Rotationsdynamik gebildet:

$$\dot{\vec{\Omega}}_{m,ref} = \tilde{\mathbf{T}}_{\vec{\Omega}} \cdot (\vec{\Omega}_{soll} - \vec{\Omega}_{m,ref}) - \nu_{h,\dot{\vec{\Omega}}}, \tag{4.92}$$

wobei die Matrix $\tilde{\mathbf{T}}_{\vec{\Omega}}$ den Zeitkonstanten der Wunschdynamik des jeweiligen Winkels entspricht,

$$\tilde{\mathbf{T}}_{\vec{\Omega}} = \begin{bmatrix} \frac{1}{T_\Phi} & 0 & 0 \\ 0 & \frac{1}{T_\Theta} & 0 \\ 0 & 0 & \frac{1}{T_\Psi} \end{bmatrix}, \tag{4.93}$$

also die Geschwindigkeit, mit dem das System einem Sollwert folgt. Dabei ist darauf zu achten, dass diese Dynamik genügend Abstand zur Rotationsdynamik aufweist, damit eine Frequenztrennung garantiert ist. So wie bei der Rotationsdynamik gilt hier der relative Grad $r = 1$, woraus ein Referenzmodell erster Ordnung resultiert. Die Lagedynamik bildet die äußere Schleife dieser Regler-Kaskade, womit in der Lagedynamik die Übertragungsbandbreite des Gesamtsystems festgelegt wird.

Die Pseudo-Steuergröße $\nu_{\dot{\vec{\Omega}}} = [\nu_{\dot{\Phi}}, \nu_{\dot{\Theta}}, \nu_{\dot{\Psi}}]^T$ bildet sich wieder nach dem Schema des Referenzmodells mit PCH und Fehlerrückführung, wobei im Gegensatz zu den bisher verwendeten Anordnungen ein zusätzlicher Integral-Anteil addiert wird, um stationäre Regelungsgenauigkeit in der Fehlerdynamik zu erreichen.

$$\nu_{\dot{\vec{\Omega}}} = \tilde{\mathbf{T}}_{\vec{\Omega}} \cdot (\vec{\Omega}_{soll} - \vec{\Omega}_{m,ref}) + \mathbf{K}_{P,\vec{\Omega}} \cdot (\vec{\Omega}_{m,ref} - \vec{\Omega}) + \mathbf{K}_{I,\vec{\Omega}} \cdot \int (\vec{\Omega}_{m,ref} - \vec{\Omega}) \cdot dt. \tag{4.94}$$

Mit den Matrizen der Zeitkonstanten der Fehlerrückführung

$$\mathbf{K}_{P,\vec{\Omega}} = \begin{bmatrix} K_{P,\Phi} & 0 & 0 \\ 0 & K_{P,\Theta} & 0 \\ 0 & 0 & K_{P,\Psi} \end{bmatrix} \quad \text{und} \quad \mathbf{K}_{I,\vec{\Omega}} = \begin{bmatrix} K_{I,\Phi} & 0 & 0 \\ 0 & K_{I,\Theta} & 0 \\ 0 & 0 & K_{I,\Psi} \end{bmatrix}, \tag{4.95}$$

kann die Fehlerdynamik beeinflusst werden. Das bedeutet, an dieser Stelle wird die Geschwindigkeit beeinflusst, mit der das System auf einen Fehler $\vec{\Delta}$ reagiert, aller-

dings nicht, wie schnell das System dem Sollwert folgt. Wie bei der Zeitkonstantenmatrix $\tilde{T}_{\vec{\Omega}}$ ist auch auf die Frequenztrennung zur Rotationsdynamik zu achten. Zusammenfassend zum Referenzmodell ist in der Abbildung 4.22 dessen Aufbau mit PCH-Eingang und Fehlerregler dargestellt.

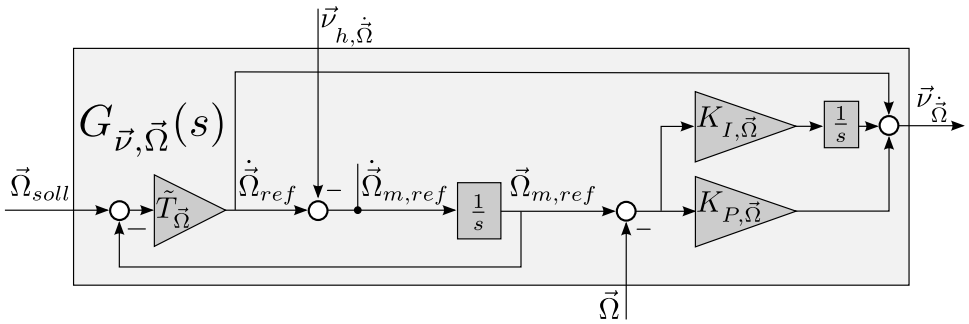


Abbildung 4.17.: Referenzmodell erster Ordnung mit Fehlerrückführung.

Der Lagefehler wird aus der Differenz zwischen gemessener Lage und Lage-Referenzsignal gebildet,

$$e_{\vec{\Omega}} = \vec{\Omega} - \vec{\Omega}_{m,ref} = \begin{bmatrix} e_\Phi \\ e_\Theta \\ e_\Psi \end{bmatrix} = \begin{bmatrix} \Phi - \Phi_{m,ref} \\ \Theta - \Theta_{m,ref} \\ \Psi - \Psi_{m,ref} \end{bmatrix} \tag{4.96}$$

und daraus resultiert die Fehlerdynamik, in der auch der Integralteil des Fehlerreglers sichtbar wird:

$$\dot{\vec{e}}_{\vec{\Omega}} + K_{P,\vec{\Omega}} \cdot \vec{e}_{\vec{\Omega}} + K_{\vec{\Omega},I} \cdot \int \vec{e}_{\vec{\Omega}} \cdot dt = \vec{\Delta}_{\vec{\Omega}}. \tag{4.97}$$

Letztendlich muss noch das Hedge-Signal $\nu_{h,\dot{\vec{\Omega}}}$ gebildet werden. Dabei wird, angelehnt an [Hol04], die Differenz zwischen den kommandierten Drehraten und den, aus der inneren Schleife resultierenden tatsächlichen Werten, als Aktuatorfehler interpretiert. Es wird also angenommen, aus Sicht des Lagereglers die innere Schleife als Stelldynamik betrachten zu können. Zur Bestimmung des Hedge-Signals werden die gemessenen Drehraten mit Gleichung (3.13) in die erreichte Pseudosteuergröße transformiert,

$$\hat{\vec{\nu}}_{\dot{\vec{\Omega}}} = \mathbf{M}_{\Phi f} \cdot \vec{\omega}, \tag{4.98}$$

womit das Hedge-Signal als Differenz der kommandierten Pseudosteuergröße $\vec{\nu}_{\dot{\vec{\Omega}}}$ und der tatsächlich erfüllbaren Pseudosteuergröße $\hat{\vec{\nu}}_{\dot{\vec{\Omega}}}$

$$\vec{\nu}_{h,\dot{\vec{\Omega}}} = \vec{\nu}_{\dot{\vec{\Omega}}} - \hat{\vec{\nu}}_{\dot{\vec{\Omega}}} \tag{4.99}$$

resultiert.

Zusammenfassend ist die Definition der Lageregelung in Abbildung 4.18 dargestellt. Der Drehratenregler aus Abbildung 4.16 ist dabei im Block „Rotationsdynamik“ enthalten. Deutlich zu erkennen ist die Lagedynamik $\mathbf{F}_{\vec{\Omega}}(\vec{x},\vec{u})$, die hinter der Rotationsdynamik steht. Im Gegensatz zu Abbildung 4.14 wird die Gesamtdynamik nicht in einem Block zusammengefasst. Damit wird noch einmal deutlich, dass die Gesamtdynamik auch beim Inversionsregler kaskadisch aufgebaut werden kann, ähnlich dem in der Flugregelung üblichen Linearregleraufbau. Gleichzeitig sei an dieser Stelle angemerkt, dass sowohl die Darstellung in getrennter Form als auch die Gesamtdynamikdarstellung möglich ist.

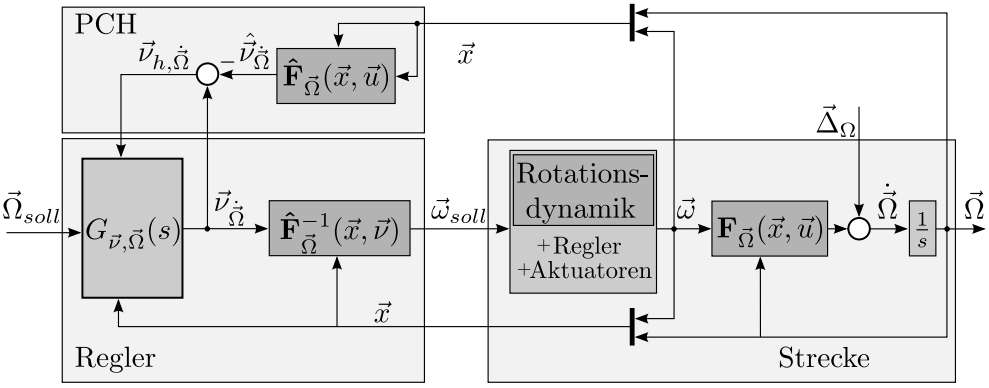


Abbildung 4.18.: Übersicht der äußeren Kaskade der Eulerinversion mit relativem Grad eins.

4.8.3 Dynamische Inversion im Flugversuch

In den letzten Abschnitten wurden bereits die Grundlagen und die Anwendung der dynamischen Inversion auf das unbemannte Luftfahrzeug T200 bzw. Twinstar beschrieben. Mit der Inbetriebnahme des Kalman Filters und Bahnreglers stehen die Komponenten für einen Flugversuch mit der dynamischen Inversion zur Verfügung. Man kann sich leicht vorstellen, dass die „perfekte“ dynamische Inversion ohne Inversionsfehler in der Simulation problemlos der Solltrajektorie bzw. den vorgegebe-

nen Winkeln folgen kann. Im realen Flugversuch ist die Situation erwartungsgemäß anders. Tatsächlich ist die bereits beschriebene Konfiguration der dynamischen Inversion ohne adaptive Elemente unter Berücksichtigung von Fehlern praktisch fluguntauglich, wie an folgendem Flugversuch beispielhaft dargestellt wird, vgl. dazu die Flugbahn in Abbildung 4.19.

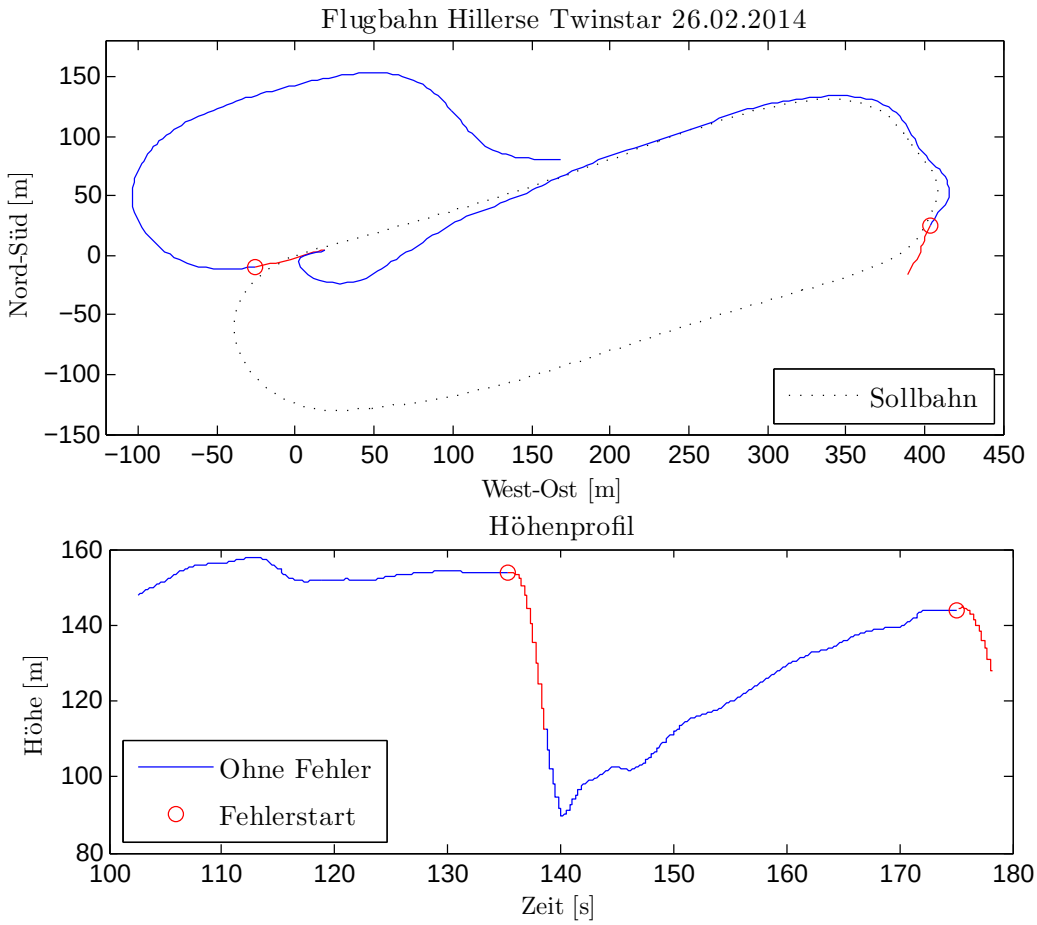


Abbildung 4.19.: Flugversuch dynamische Inversion; GPS-Bahnverlauf und Höhe; Trimmfehler führen zum Absturz.

Das Ergebnis des Flugversuches kann folgend beschrieben werden: Mit dem Einschalten des Inversionsreglers zum Zeitpunkt [$t = 135\ s$] beschreibt die Flughöhe des Flugzeugs einen steilen Sinkflug mit immer weiter sinkendem Nickwinkel, wie in Abbildung 4.20 dargestellt. Innerhalb weniger Sekunden muss das Experiment vom Piloten bei einem Nickwinkel von 90° und einem Kurs Richtung Süd-West abgebrochen werden. Nach einem Steigflug auf eine sichere Höhe wird das Experiment bei [t=175 s] wiederholt und resultiert in gleichem Verhalten.

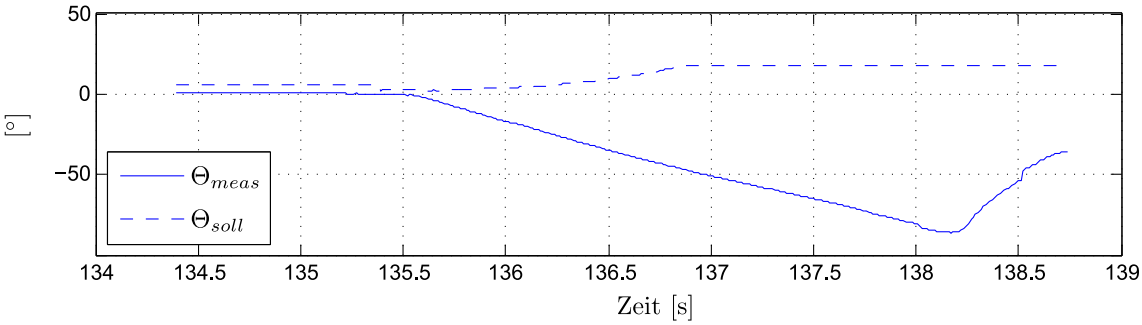


Abbildung 4.20.: Flugversuch; gemessene und kommandierte Nicklage; Trimmfehler führen zum Absturz.

Letztendlich führen Trimmfehler im Höhenruder zu diesem Verhalten, wie in der Simulation in Schaubild 4.21 nachgebildet werden kann. Hierbei wird beispielhaft ein Trimmfehler im Höhenruder von $\Theta = -10°$ abgebildet.

Die Trimmung im Flugversuch ist mit dieser Reglerkonfiguration nur schwer durchzuführen. Betrachtet man noch einmal den Aufbau der Inversionsreglerkaskade aus Abbildung 4.18, so wird die Ursache dieser Problematik deutlich. Wie bereits erwähnt, ist das Ziel des PCH, den Aktuator für den Regler „unsichtbar“ zu machen. Im Fall des Lagereglers wird die Drehratenregelung als Aktuator betrachtet. Bildlich gesehen kann man sagen, der Lageregler „ignoriert“ den aus einer fehlerhaften Drehratenregelung entstandenen Fehler in der Lage. Die Konfiguration des PCH führt somit zu einer unerwünschten Fehlerrückführung der fehlerhaft geregelten Drehraten $\vec{\omega}_{meas}$ und im Integrator des Lage-Referenzmodells werden Drehratenfehler zu einer fehlerhaften Referenzlage $\vec{\Omega}_{ref}$ aufintegriert. Ohne adaptive Elemente, die den Fehler in der Drehratenregelung kompensieren, ist diese Reglerstruktur impraktikabel.

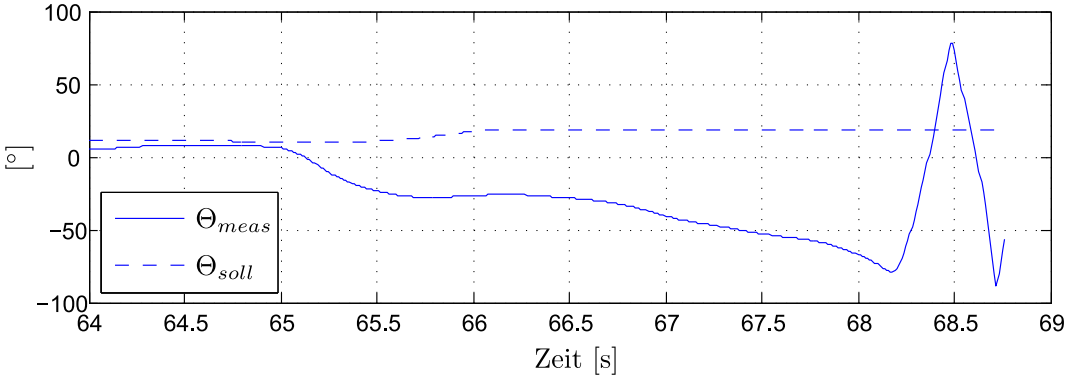


Abbildung 4.21.: Simulation; gemessene und kommandierte Nicklage; Trimmfehler führen zum Absturz.

4.9 Reference-Control-Hedging

Wie im letzten Flugversuch dargestellt, ist die bisherige Konfiguration für einen Flugversuch mit Inversionsfehlern ungeeignet. Um dennoch einen Eindruck vermitteln zu können, wie sich die dynamische Inversion ohne adaptive Elemente im Flugversuch verhält, wird zu diesem Zweck eine leicht modifizierte Version des PCH vorgestellt. Die Idee des PCH basiert darauf, entweder die erreichbare Sollgröße eines Aktuators in die übergeordnete Reglerschleife zurückzuführen, z.B. mit einem Aktuatormodell, oder die tatsächlich erreichte Stellgröße des Aktuators, sofern die Stellgröße gemessen werden kann. Die Problematik soll einleitend mit der Abbildung 4.22 an einem Referenzmodell, wie es in der Lageregelung verwendet wird und einem stark vereinfachten Aktuator, dargestellt werden.

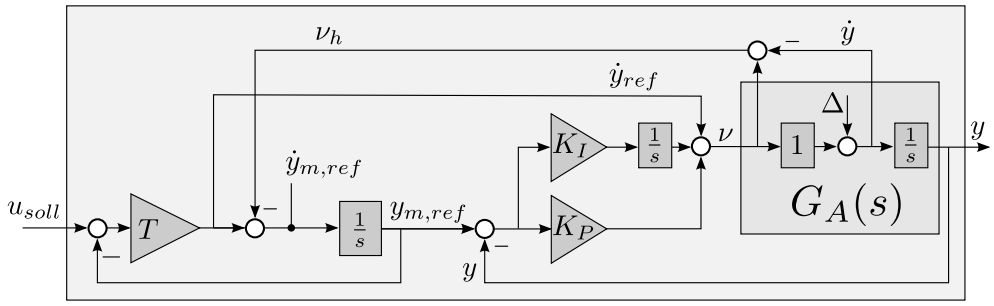


Abbildung 4.22.: Referenzmodell erster Ordnung mit Fehlerregler und einem stark vereinfachten Aktuator.

Wird ein Aktuatormodell verwendet, sind Aktuatorfehler unkritisch, da dem Fehlerregler die Möglichkeit gegeben wird, die Aktuatorfehler zu kompensieren. Wird dagegen das gemessene Stellsignal des Aktuators zur Bildung der Hedge-Größe herangezogen, wie oben dargestellt, führen Fehler im Aktuator zum Ignorieren dieses Fehlers im Fehlerregler. Wie bereits gezeigt wurde, ist der Effekt des „Ignorierens“ im Fall von Aktuatorbegrenzungen gewünscht. Die Rückführung des Aktuatorausgangs bildet allerdings bei einem fehlerhaften Aktuator eine Rückführung der Fehlergröße in das Referenzmodell und damit in den Fehlerregler. Die stark vereinfachte Struktur des „Aktuators“ $G_A(s)$ in Abbildung 4.22 erlaubt eine Betrachtung des Systemverhaltens im Bildbereich. Der Ausgang $y(s)$ des Systems bildet sich im Bildbereich nach folgender Gleichung:

$$y(s) = \frac{\Delta}{s} + \frac{1}{s}\left[\frac{K_P s + K_I}{s}\left(y_{m,ref} - y\right) + \dot{y}_{ref}\right], \tag{4.100}$$

wobei die Ableitung des Referenzausgangs im Messpunkt m folgendem Zusammenhang folgt:

$$\begin{aligned}\dot{y}_{m,ref}(s) &= T\left(u_{soll} - y_{m,ref}\right) - \nu_h \\ &= T\left(u_{soll} - y_{m,ref}\right) + \Delta\end{aligned} \tag{4.101}$$

und der Ausgang des Referenzmodells in Abhängigkeit von der Stellgröße dargestellt werden kann:

$$y_{m,ref}(s) = \frac{T}{s+T}\left(u_{soll} + \frac{\Delta}{T}\right). \tag{4.102}$$

Damit kann der Ausgang der Gleichung (4.100) in Abhängigkeit vom Fehler Δ und von der Stellgröße u_{soll} formuliert werden:

$$y(s) = \frac{s}{s^2 + K_P s + K_I}\left[\frac{T(K_P - T)s + TK_I}{s(s+T)}\left(u_{soll} + \frac{\Delta}{T}\right) + Tu_{soll} + \Delta\right]. \tag{4.103}$$

Die Sprungantwort $u_{soll}(s) = 1/s, \Delta(s) = 1/s$ kann so, mit der aus der Regelungstechnik üblichen Laplace Transformation, berechnet werden:

$$\lim_{t\to\infty} y(t) = \lim_{s\to 0} y(s)\cdot s = u_{soll} + \frac{\Delta}{T}. \tag{4.104}$$

Überträgt man diese Erkenntnisse auf den Lageregler in Kapitel 4.8.2, so wird deutlich, dass auch mit einem Integrator im Fehlerregler der Lage, ein Fehler der Rotationsdynamik nicht kompensiert wird. In der Konfiguration des PCH der Lageregelung wird die Drehratenregelung als „Aktuator“ interpretiert und dessen Ausgang, also die gemessene Drehrate $\vec{\omega}$, zur Bildung des Hedge-Signals $\hat{\vec{\nu}}_{\dot{\vec{\Omega}}}$ verwendet, vgl. Gleichung (4.98). Eine Alternative zur gemessenen Drehrate bildet das hier vorgestellte Reference-Control-Hedging (RCH), mit der Verwendung von $\vec{\omega}_{m,ref}$, sodass sich für die erreichbare Pseudosteuergröße der Lageregelung folgender Zusammenhang ergibt:

$$\hat{\vec{\nu}}_{\dot{\vec{\Omega}}} = \mathbf{M}_{\Phi f}\cdot\vec{\omega}_{m,ref}. \tag{4.105}$$

Im Messpunkt $\vec{\omega}_{m,ref}$ (vgl. Grafik 4.15) des Rotationsdynamik-Referenzmodells werden neben dessen Geschwindigkeit auch die Begrenzungen der wahren Aktuatoren berücksichtigt. Die Berücksichtigung der wahren Aktuatoren der Drehratenregelung, hier Querruder und Höhenruder, wird beim Betrachten von Gleichung (4.74) im Messpunkt $\vec{\omega}_{m,ref}$ deutlich. Die Referenzgröße $\vec{\omega}_{m,ref}$ entspricht somit den, nach dem

Referenzmodell der Rotationsdynamik, erreichbaren Drehraten. Eine Rückführung der gemessenen Drehraten wird offensichtlich umgangen. Diese RCH-Variation hat also den Vorteil, dass Fehler in der Drehratenregelung nicht mehr in das Referenzmodell der Lageregelung zurückgeführt werden. Die Änderung der Reglerstruktur wird in der Grafik 4.23 dargestellt. Erste Untersuchungen zeigen zudem, dass im Gegensatz zum PCH, mit dem RCH der Fehlerregler in der Lageregelung deutlich schneller gewählt werden kann.

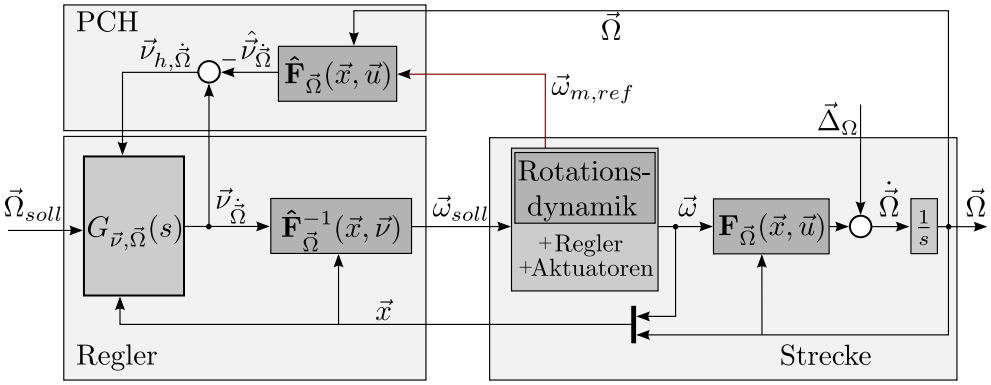


Abbildung 4.23.: Übersicht der äußeren Kaskade der Eulerinversion mit relativem Grad eins und RCH.

Ein Integrator im Fehlerregler der Rotationsdynamik ist zwar auch in der Lage, z.B. die Trimmfehler der Rotationsdynamik zu kompensieren und eine Übertragung von Inversionsfehlern in die Lageregelung zu minimieren. Der Vorteil des RCH gegenüber dem PCH liegt allerdings darin, dass ein Vergleich zwischen der vorgeschlagenen Struktur der adaptiven dynamischen Inversion in [Hol04, Krü12] und einer dynamischen Inversion ohne adaptive Elemente stattfinden kann. Während im nächsten Flugversuch die RCH-Variation zum Einsatz kommt, wird im Fall der später vorgestellten adaptiven Elemente weiter das PCH verwendet. Eine Kombination von RCH und der dynamischen Inversion mit adaptiven Elementen ist allerdings auch denkbar.

4.9.1 Dynamische Inversion mit RCH in der Simulation

Mit der RCH-Variation wird, im Rahmen dieser Arbeit, erstmals eine sinnvolle Darstellung der Folgen von Inversionsfehlern in Simulationen bzw. Flugversuchen der dynamischen Inversion möglich. Das Ziel der folgenden Darstellung ist, dem Leser zu vermitteln, wie die bisherige Regler-Konfiguration im Fall von Inversions-

fehlern reagiert. Da der Fokus auf der späteren Untersuchung der adaptiven Elemente im Rahmen einer suboptimalen Reglerkonfiguration liegt, sei an dieser Stelle nur erwähnt, dass eine optimierte Parameterkonfiguration der Lageregelung zu deutlich verbesserten Ergebnissen führt. Ein mögliches Szenario zur Demonstration der RCH-Eigenschaften wird im Bahnverlauf anhand von Grafik 4.24 dargestellt. Hierfür wird der bereits im Flugversuch des Basisreglers erwähnte Querruderfehler

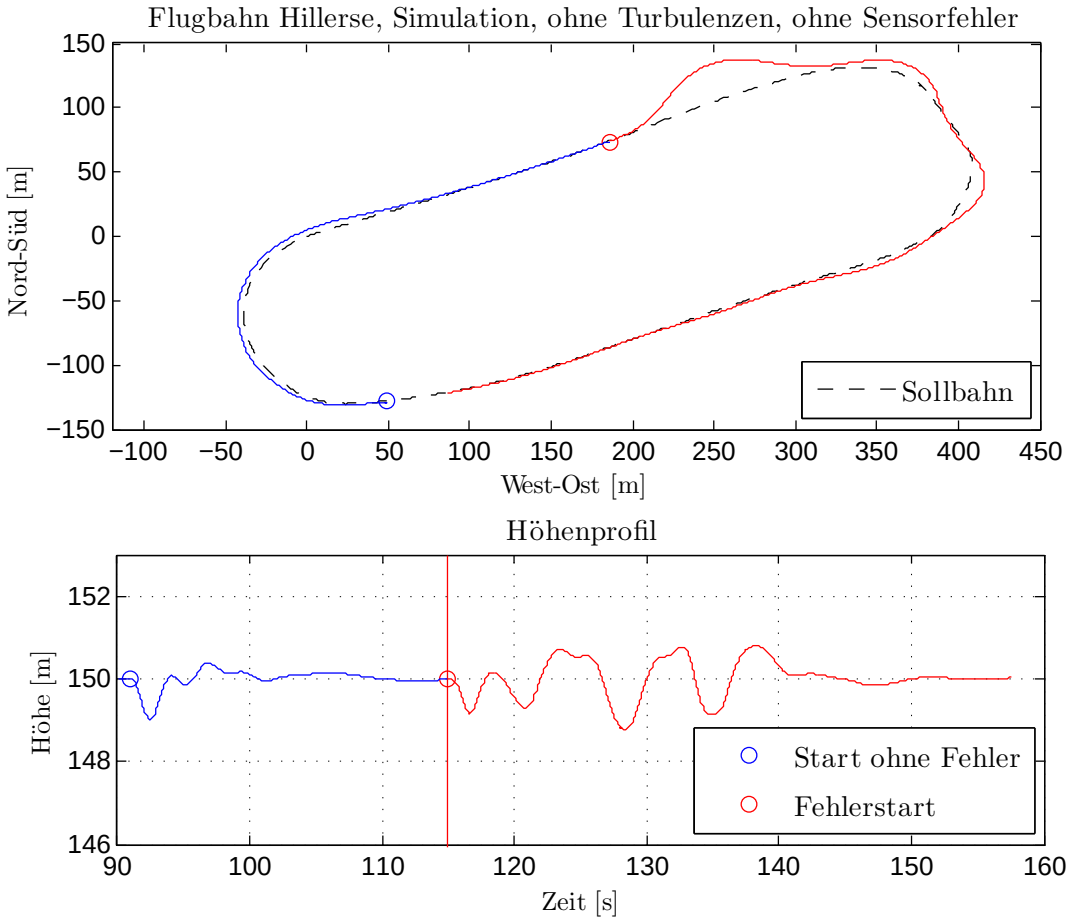


Abbildung 4.24.: Simulation Bahn- und Höhenverlauf; dynamische Inversion mit RCH; blauer Kreis: Inversion Start; roter Kreis: Fehlerstart (blockiertes linkes Querruder $\xi_{li} = 7°$).

simuliert. Während mit PCH-Konfiguration bereits bei Trimmfehlern eine Bahnfolge nicht mehr möglich ist, geschweige denn ein Einfrieren des Querruders, ist ein Weiterfliegen mit der RCH-Konfiguration immer noch realisierbar. Vor dem Deaktivieren des Querruders bei $\xi_{li} = 7°$ ist das Flugzeug offensichtlich gut in der Lage, der Sollbahn zu folgen. Nach der Aktivierung des Fehlers ist ein Ausbrechen vom gewünschten Bahnverlauf zu beobachten, ein Weiterfliegen ist in dieser Simulation dennoch möglich.

Der Unterschied zwischen idealer Inversion und Inversionsfehler ist ebenfalls im Verlauf der Lagewinkel zu erkennen, wie in Grafik 4.25 und 4.26 dargestellt. Ab dem Zeitpunkt der Fehlerzuschaltung [t=115 s] führt die sprungartige Veränderung der linken Querruderstellung zu einem kurzfristigen Ausbrechen des gemessenen Rollwinkels. Der Rolllageregler braucht ca. 5 s um den Fehler abzubauen. Man kann sich anhand dieses Verlaufs leicht vorstellen, dass der Rollratenregler nicht mehr optimal an die veränderte Rolldynamik angepasst ist. Zwar ist die Nickdynamik in diesem Versuch unverändert, durch die Schwankungen der Rolllage ab dem Fehlerzeitpunkt kommt es aber auch im Nicklageverlauf zu Abweichungen.

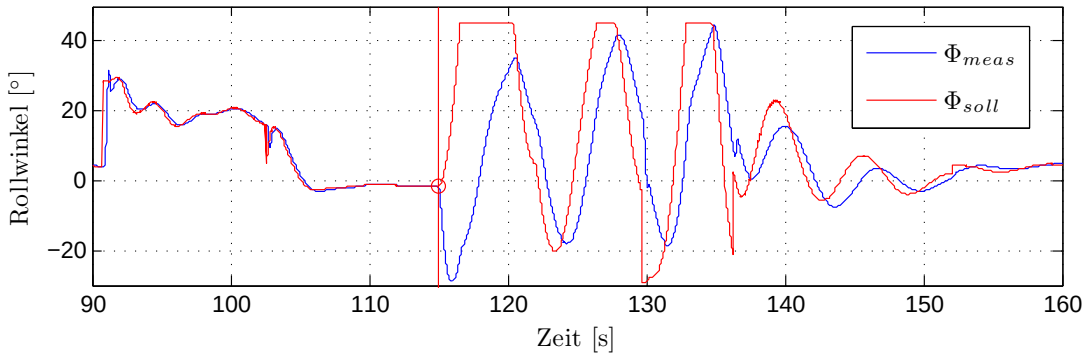


Abbildung 4.25.: Simulation Rollwinkel; dynamische Inversion mit RCH; roter Kreis: Fehlerstart (blockiertes linkes Querruder $\xi_{li} = 7°$).

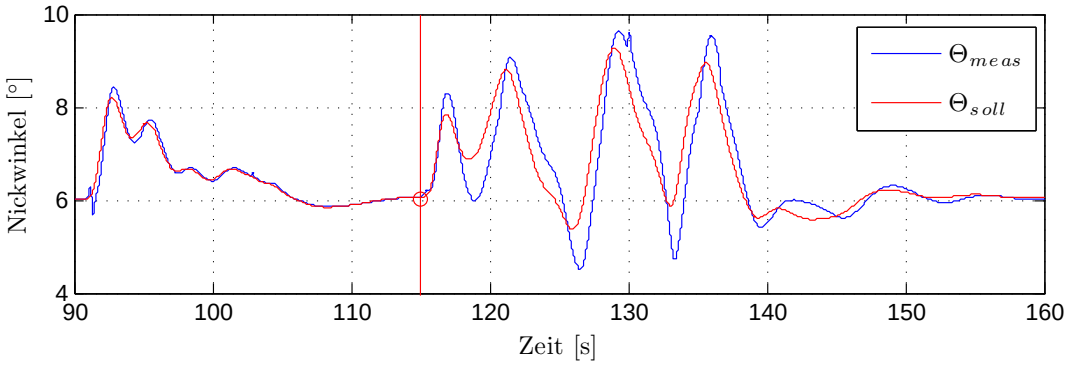


Abbildung 4.26.: Simulation Nickwinkel; dynamische Inversion mit RCH; roter Kreis: Fehlerstart (blockiertes linkes Querruder $\xi_{li} = 7°$).

Betrachtet man die Rollrate in Abbildung 4.27, so wird deutlich, dass mit der gewählten Reglerstruktur eine bleibende Regelabweichung in der inneren Kaskade entsteht. Der Regelfehler e_p in der Rollrate wird nicht abgebaut. Zusammenfassend prognostiziert die Simulation eine aus einer veränderten Flugzeugdynamik resultierende,

verschlechterte Reglergüte und durch den Sprung im Querruder verursachte Abweichungen zwischen kommandierten und gemessenen Zustandsgrößen.

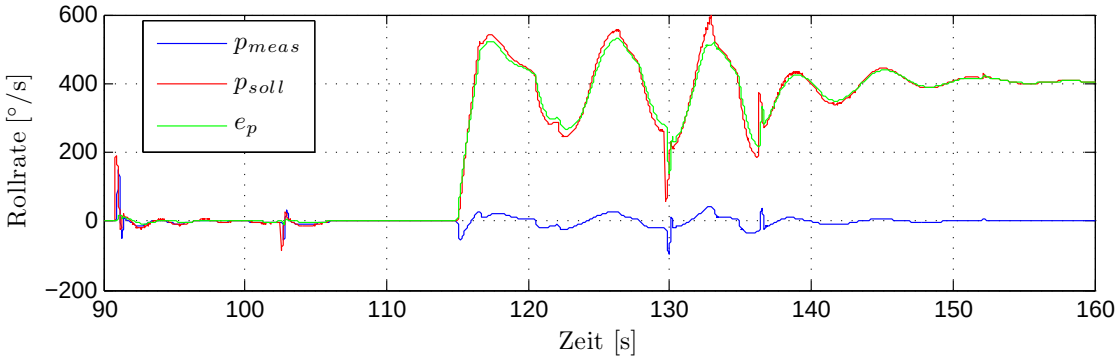


Abbildung 4.27.: Simulation; Rollrate, Rollratenkommando und Regelfehler; blockiertes linkes Querruder $\xi_{li} = 7°$ ab [t=115 s].

4.9.2 Dynamische Inversion mit RCH im Flugversuch

Bereits in der Simulation konnten Einflüsse von System-Degradationen, in Form einer Querruderblockade, auf die dynamische Inversion beobachtet werden. Diese Inversionsfehler führten zu Abweichungen der Soll- und Istgrößen. Zur weiteren Darstellung der Reglereigenschaften wird das Simulationsexperiment nun im realen Flugversuch wiederholt (vlg. dazu Grafik 4.28 bis 4.31).

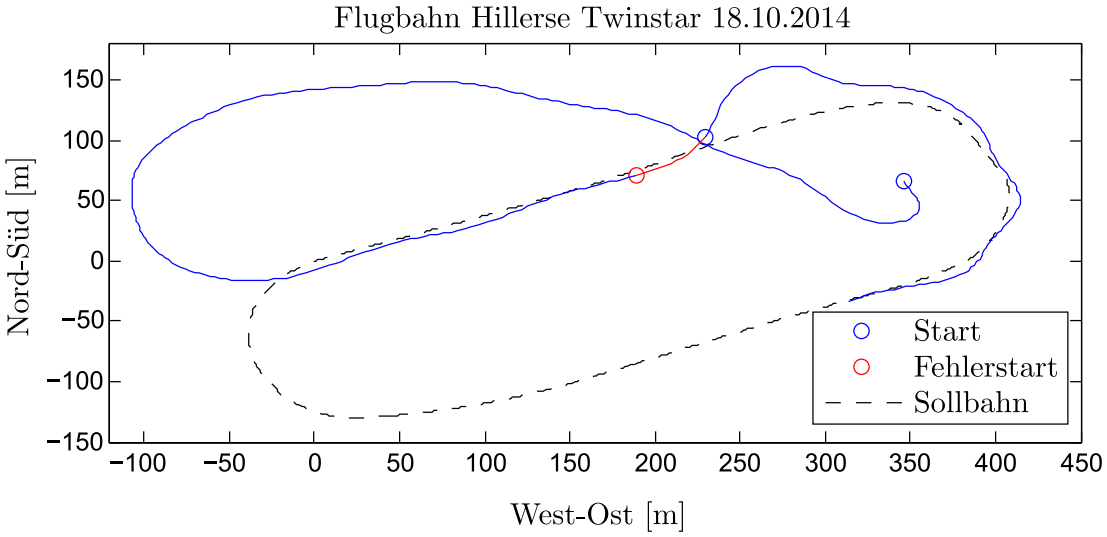


Abbildung 4.28.: Flugversuch dynamische Inversion mit RCH; GPS-Bahnverlauf; blau: ohne Fehler; rot: mit blockiertem linken Querruder $\xi_{li} = 7°$.

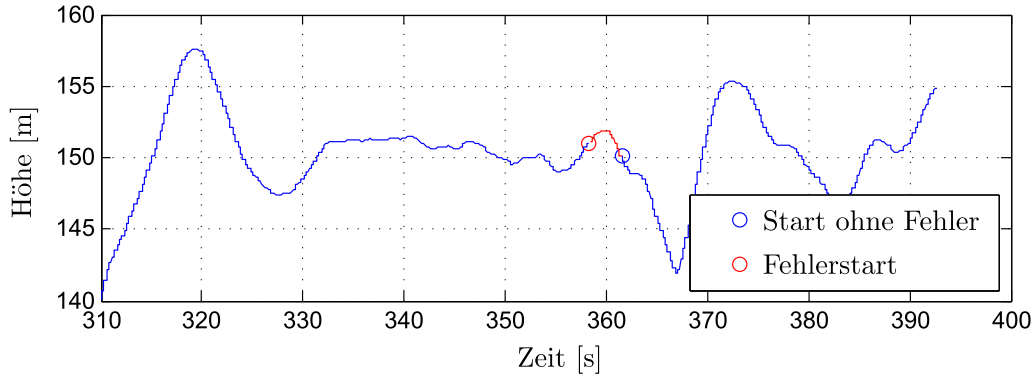


Abbildung 4.29.: Flugversuch dynamische Inversion mit RCH; Höhenverlauf; blau: ohne Fehler; rot: mit blockiertem linken Querruder $\xi_{li} = 7°$.

Der Flugversuch wird wieder mit demselben Schema wie die vorangehenden Versuche durchgeführt. Nach einer gewissen Zeit wird die bereits erwähnte Querruderblockade ξ_{li} zugeschaltet. Ausgenommen der Parameter Bezugsfläche S, Masse m und dem Derivat $C_{L,\xi}$, wird das gesamte Modell des T200 übernommen. Da der Inversionsregler mit dem Modell und der Inversion des T200 Flugzeugs erstellt wird, der Flugversuch allerdings mit der Twinstar stattfindet, sind Bahn- und Lageverlaufabweichungen durch Inversionsfehler zu erwarten. Ohne zu weit vorzugreifen, soll bereits erwähnt werden, dass dieser Flugversuch mit gleichen Parametern, allerdings mit Zuschaltung von adaptiven Elementen fortgeführt wird.

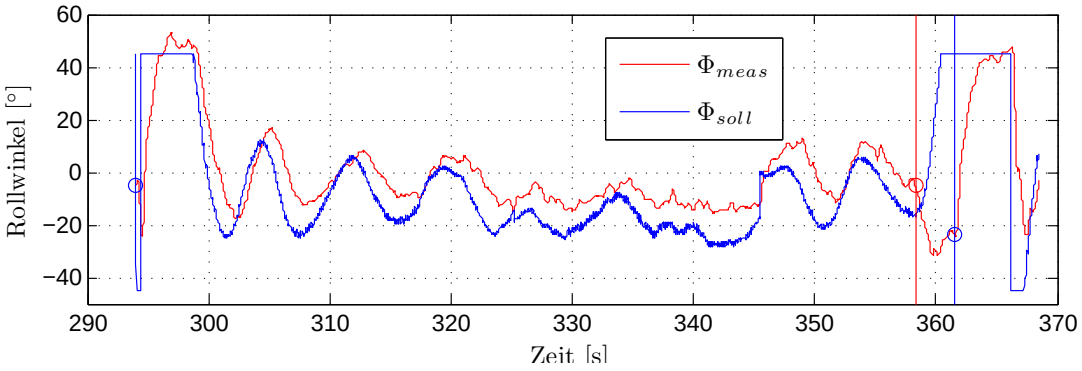


Abbildung 4.30.: Rollwinkel; dynamische Inversion mit RCH; blauer Kreis: Inversion Start; rote vert. Linie: Fehlerstart (blockiertes linkes Querruder $\xi_{li} = 7°$); blaue vert. Linie: Fehler abgeschaltet. $\xi_{li} = 7°$).

Während eine Bahnführung bis zum Einschaltzeitpunkt des Querruderfehlers möglich ist, wurde in diesem Flugexperiment das Einfrieren des linken Querruders aus Sicherheitsgründen nach wenigen Sekunden abgebrochen. Der Einfluss von Inversi-

onsfehlern wird auch bei Betrachtung der Verläufe von Roll- und Nickwinkel deutlich, die in Grafik 4.30 und 4.31 abgebildet werden. Der zugeschaltete Querruderfehler führt, ähnlich wie in der Simulation im letzten Abschnitt, zu einem Ausbrechen des Rollwinkels.

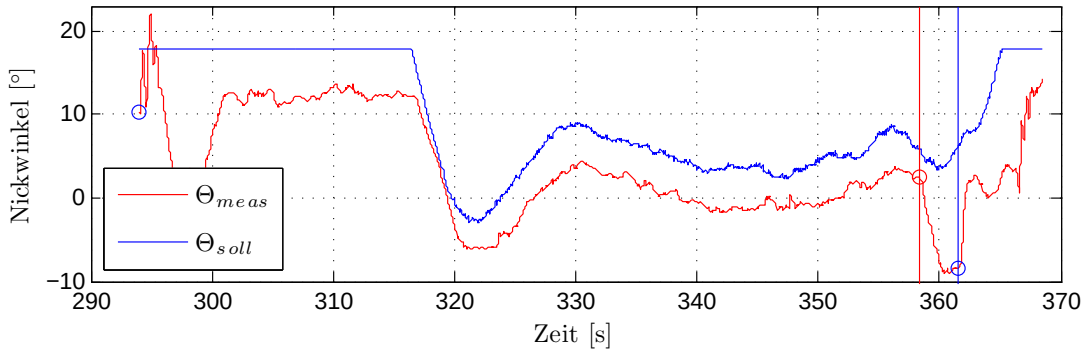


Abbildung 4.31.: Nickwinkel; dynamische Inversion mit RCH; blauer Kreis: Inversion Start; rote vert. Linie: Fehlerstart (blockiertes linkes Querruder $\xi_{li} = 7°$); blaue vert. Linie: Fehler abgeschaltet.

Zusammenfassend kann man sagen, dass die durch Inversionsfehler zu erwartenden, nicht optimalen Bahn- und Lageverläufe im Flugversuch bestätigt werden können. Ein zusätzlicher Querruderfehler führt zudem zu weiteren Fehlern in Bahn- und Lageverlauf.

5 Neuronale Netze

Wie bereits gezeigt werden konnte, können immer wieder Situationen auftreten, in denen lineare Regler oder das Konzept der dynamischen Inversion an seine Grenzen stößt. Adaptive Glieder können in solchen Momenten sehr schnell auf die Veränderungen im System reagieren und auf diese Weise die Stabilität wieder herstellen. Aus den aufgeführten Gründen werden im Folgenden die künstlichen neuronale Netze als adaptive Elemente in der dynamischen Inversion vorgestellt.

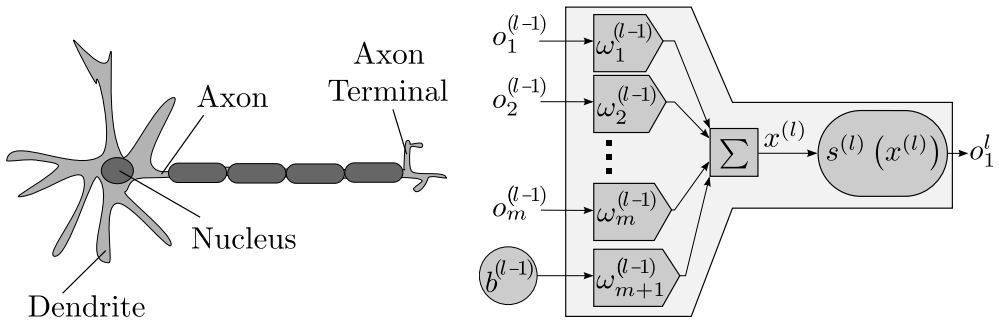
Nach [SH89] können neuronale Netze als universelle Approximatoren betrachtet werden und sind mit ihren Echtzeitfähigkeiten besonders für regelungstechnische Anwendungen, in denen Adaptivität gefordert wird, geeignet. Das Spektrum der möglichen Anwendungen reicht von einfachen mathematischen Modellen, z.B. dem Erlernen einfacher logischer Verknüpfungen wie XOR, AND, OR, bis hin zu komplexen Simulationen von ganzen Hirnregionen [RA13]. Ein gutes Grundlagenwerk bietet [Roj96], dabei werden verschiedene neuronale Konzepte in vielen Variationen vorgestellt. Mit den Untersuchungen in [Lan13, Krü12] wurde eine Netzkonfiguration gewählt, die im Hinblick auf die bevorstehenden Aufgaben besonders geeignet erscheint. Dabei spielen Faktoren wie CPU-Ressourcen und Adaptions-Güte eine wesentliche Rolle.

Im Folgenden werden die Grundlagen der verwendeten Algorithmen ausgeführt. Die Algorithmen werden dabei auch immer wieder auf ihre Echtzeitfähigkeit hin untersucht, um im Rahmen der dynamischen Inversion Anwendung im Flugversuch zu finden.

5.1 Das künstliche Neuron

Einleitend zu den neuronalen Netzen wird zuerst das künstliche Neuron vorgestellt, das die Basis der vorgestellten Netze bildet. Das künstliche Neuron ist eine starke Vereinfachung des biologischen Neurons. Beim natürlichen Vorbild beginnt der Ausgang des Neurons, das sogenannte Axon, zu feuern, sobald die Summe der Eingangssignale an den Dendriten einen gewissen Schwellwert überschreitet. Wie beim natürlichen Vorbild (vgl. Grafik 6.14(a)) wird das Ausgangssignal des künstlichen Neurons aus den gewichteten Eingängen des Neurons gebildet. Der Schwellwert des „Feuerns“ wird mit einer sogenannten Aktivierungsfunktion s simuliert.

Die Zusammenhänge können wie folgt formuliert werden: Im vereinfachten Neuronenmodell (vgl. Grafik 6.14(b)) werden hierzu, für ein i-tes Neuron $o_i^{(l)}$ einer Schicht l, die Neuronenausgänge $o_j^{(l-1)}$ der vorangehenden Schicht $(l-1)$ mit den Gewichten $w_{i,j}^{(l-1)}$ multipliziert und die Summe in der Aktivierungsfunktion $s^{(l)}\left(x^{(l)}\right)$ verarbei-



(a) Schema einer Nervenzelle. (b) Das vereinfachte Neuronenmodell.

Abbildung 5.1.: Vergleich von Neuronen.

tet. Da jede Schicht ein Bias $b^{(l-1)}$ als zusätzlichen „virtuellen“ Eingang besitzt, darf dieses nicht unberücksichtigt bleiben. Statt den Bias im späteren Trainingsverfahren als variablen Parameter zu verändern, kann dazu das nachgestellte Gewicht $w_{i,m+1}^{(l-1)}$ variiert werden. So ergibt sich der Ausgang des Neurons zu

$$o_i^{(l)} = s^{(l)}\left(x_i^{(l)}\right), \tag{5.1}$$

mit dem Eingang $x_i^{(l)}$ der Aktivierungsfunktion,

$$x_i^{(l)} = \sum_{j=1}^{m}\left(o_j^{(l-1)} \cdot w_{i,j}^{(l-1)}\right) + b^{(l-1)} \cdot w_{i,m+1}^{(l-1)}. \tag{5.2}$$

Um die folgenden Berechnungen zu vereinfachen, wird der Bias als statischer „virtueller“ Ausgang $o_m^{(l-1)} = b^{(l-1)}$ betrachtet. So kann Gleichung (5.2) vereinfacht werden zu:

$$x_i^{(l)} = \sum_{j=1}^{m}\left(o_j^{(l-1)} \cdot w_{i,j}^{(l-1)}\right). \tag{5.3}$$

Im biologischen Vorbild variiert die Aktivierungsfunktion je nach Nervenzellentyp. Prinzipiell können diese komplexen elektro-chemischen Mechanismen bis auf molekularer Ebene nachgebildet werden, z.B. [CB05]. Das Ziel dieser Anwendung liegt aber nicht in einer getreuen Nachbildung, sondern in einer optimalen Nutzung der adaptiven Eigenschaften. In Regelungsanwendungen werden dazu auf Grund von

Echtzeitfähigkeit, mathematischer Nachweisbarkeit von Stabilität und Implementierbarkeit stark vereinfachte mathematische Funktionen verwendet. Besonders lineare sowie sigmoide Aktivierungsfunktionen haben sich bewährt [Krü12]. Denkbar wäre auch eine Sprungfunktion, diese wirkt sich aber in der späteren Anwendung wegen der fehlenden stetigen Ableitbarkeit unvorteilhaft aus.

Für die lineare Aktivierungsfunktion gilt (vgl. Grafik 5.2(a)):

$$s(x) = x. \tag{5.4}$$

Als Sigmoidfunktion wird ausschließlich die Tangens-Hyperbolicus-Funktion verwendet (vgl. Grafik 5.2(b)):

$$s(x) = \frac{2}{1 + e^{-2x}} - 1. \tag{5.5}$$

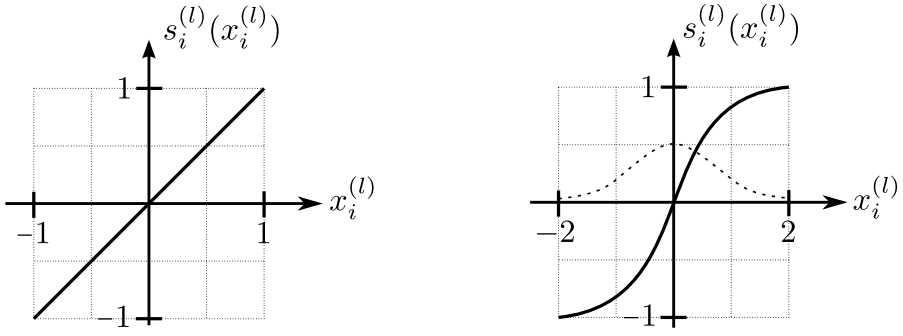
Die Sigmoidfunktion hat den Vorteil, dass die Ableitung dieser ein Abbild ihrer selbst beinhaltet,

$$\frac{\partial s(x)}{\partial x} = 1 - s(x)^2, \tag{5.6}$$

kann aber auch in ausgeschriebener Form dargestellt werden:

$$\frac{\partial s(x)}{\partial x} = \frac{4e^{-2x}}{(1 + e^{-2x})^2}. \tag{5.7}$$

Dieser Vorteil kann im folgend vorgestellten Lernverfahren im Sinne der Rechenressourcen für einen besonders effektiven Algorithmus genutzt werden.



(a) Lineare Aktivierungsfunktion.

(b) Sigmoide Aktivierungsfunktion und dessen Ableitung.

Abbildung 5.2.: Lineare und sigmoide Aktivierungsfunktion.

5.2 Aufbau neuronaler Netzwerke

Grundsätzlich ist jede Topologie, d.h. Anordnung der Neuronen möglich, jedoch ist es nötig, die verwendete Topologie genau zu definieren. Im Rahmen dieser Arbeit sind die neuronalen Netze in Schichten organisiert [Roj96]. Mit Hinblick auf die anstehenden Aufgaben haben sich besonders dreischichtige Netze bewährt [Lan13, Krü12, Möß09], wie in Grafik 5.3 dargestellt. Das Netz ist aufgebaut aus einer Eingangsschicht ($l = 1$), einer verdeckten Schicht ($l = 2$) und einer Ausgangsschicht ($l = 3$). Die Verbindungen zwischen den einzelnen Neuronen $o_j^{(l-1)}$ und $o_i^{(l)}$ entsprechen den Gewichten $w_{ij}^{(l-1)}$. Diese Propagierung der Eingangssignale $\vec{x}$ zum Ausgang y wird in der Fachliteratur auch als „Feed-Forward“ bezeichnet.

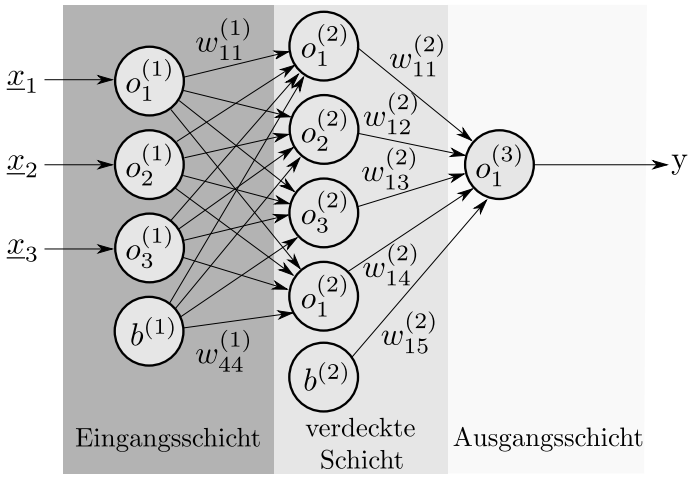


Abbildung 5.3.: Das künstliche neuronale Netz mit drei Schichten.

Im Sinne einer effizienten Formalisierung werden die Gewichte in Matrizen organisiert. Da für die anstehenden Aufgaben nur Netze mit einem Ausgang benötigt werden, reduziert sich die Matrix $\mathbf{w}^{(2)}$ auf einen Zeilenvektor.

$$\mathbf{w}^{(1)} = \begin{bmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} & \dots & w_{1,m}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,1}^{(1)} & w_{n,2}^{(1)} & \dots & w_{n,m}^{(1)} \end{bmatrix}, \mathbf{w}^{(2)} = \left[w_{1,1}^{(2)}, w_{1,2}^{(2)}, \dots, w_{1,m}^{(2)} \right]. \tag{5.8}$$

Die Indizes n entsprechen der Anzahl von Neuronen einer Schicht (l), sowie m der Neuronenanzahl einer vorangehenden Schicht ($l-1$). Der Vektor $\mathbf{w}^{(2)}$ kann aber bei Bedarf einfach auf mehrere Ausgänge, analog zur Matrix $\mathbf{w}^{(1)}$ der ersten Schicht, erweitert werden.

Die beiden Gewichtsmatrizen können auch in einer Gesamtgewichtsmatrix $\mathbf{w}$ zusammengefasst werden:

$$\mathbf{w} = \begin{bmatrix} \mathbf{w}^{(2)} & 0 \\ 0 & \mathbf{w}^{(1)} \end{bmatrix}. \tag{5.9}$$

Für das dreischichtige Netz ist mit den Gewichtsmatrizen eine besonders übersichtliche Darstellung des Ausgangs y möglich:

$$y = s^{(3)} \left(\mathbf{w}^{(2)} \cdot \vec{s}^{\,(2)} (\mathbf{w}^{(1)} \cdot \vec{s}^{\,(1)} (\underline{\vec{x}})) \right). \tag{5.10}$$

Dabei ist ersichtlich, dass sich der Ausgang durch Hintereinanderschaltung der einzelnen Schichten aus dem Eingangsvektor $\underline{\vec{x}} = [\underline{x}_1, \underline{x}_2, ..., \underline{x}_m]^T$ mit m Eingängen ergibt.

Die Aktivierungsfunktionen variieren je nach Schicht. Für die Eingangsschicht und die Ausgangsschicht hat sich die lineare Aktivierungsfunktion (5.4) etabliert, während die sigmoide Aktivierungsfunktion (5.5) Anwendung in der verdeckten Schicht findet [Krü12]. Damit kann Gleichung (5.11) weiter vereinfacht werden:

$$y = s^{(3)} \left(\mathbf{w}^{(2)} \cdot \vec{s}^{\,(2)} (\mathbf{w}^{(1)} \cdot \underline{\vec{x}}) \right). \tag{5.11}$$

5.3 Lernen mit Backpropagation

In den letzten Abschnitten wurden die neuronalen Netze in ihrer Struktur definiert und mit der Feed-Forward-Propagation können diese bereits ein Ausgangssignal aus einem Eingangssignal bilden. Ohne ein Lernverfahren besitzen diese Netze aber noch keine adaptiven Eigenschaften. Grundsätzlich basiert das Lernen auf der Rückpropagation eines Fehlers, worauf die Namensgebung des hierfür im Allgemeinen verwendeten Begriffs „Backpropagation“ zurückzuführen ist. Anhand von Grafik 5.4 kann das Prinzip veranschaulicht werden. Dabei kann man sagen, dass das Netz „rückwärts“ durchlaufen wird. Offensichtlich führt eine Änderung der Gewichte $w_{ij}^{(l)}$ zu einer Änderung des Netzausgangs y und damit zu einer Änderung des Fehlers E. Auf der anderen Seite bedeutet dies, dass ein vorhandener Fehler mit einer Änderung der Gewichte verändert werden kann. Allen Lernverfahren ist somit die Minimierung eines Netzwerkfehlers gemein, insofern, dass die Netzgewichte $w_{ij}^{(l)}$ verändert werden

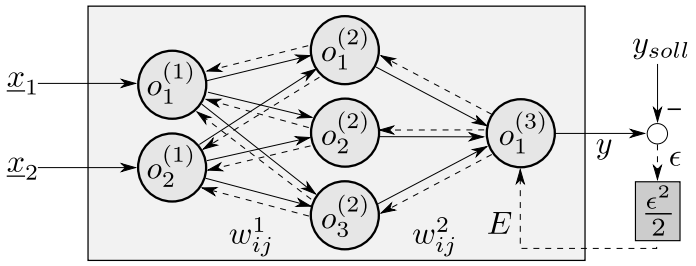


Abbildung 5.4.: Bei der Backpropagation wird das Netz „rückwärts“ durchlaufen.

müssen, um dieses Minimum zu erreichen. Die Redewendung „aus Fehlern lernen“ kann dabei als zutreffend eingestuft werden.

Als Netzfehler E wird vorerst die quadratische Fehlerfunktion zwischen dem aktuellen Ausgang y und dem gewünschten Ausgang y_{soll} zu

$$E = \frac{1}{2}(y_{soll} - y)^2 \tag{5.12}$$

festgelegt. Als Ausgabefehler ϵ wird die Abweichung zwischen dem aktuellen Ausgang y und dem gewünschten Ausgang y_{soll} definiert:

$$\epsilon = y_{soll} - y. \tag{5.13}$$

Wie sich in später vorgestellten Lernverfahren zeigen wird, sind auch andere Fehlerfunktionen zum Netzwerktraining geeignet.

5.3.1 Gradientenabstiegs-Lernverfahren

Die Problemstellung des Lernens besteht also darin, zuerst einen Zusammenhang zwischen der Fehleränderung und einer fehlerminimierenden Gewichtsänderung zu finden, um diese für die Berechnung von neuen Gewichten nutzen zu können.

Eines der ersten bekannten und effektiven Lernverfahren ist das Gradientenabstiegs-Lernverfahren, das die Grundlage der Netzwerktrainings durch Fehlerrückführung bildet. Auch das später vorgestellte Sliding-Mode-Lernverfahren bedient sich in Teilen den hier vorgestellten Mechanismen. Die ersten Formulierungen des Gradientenabstiegs-Verfahrens für neuronale Netzwerke fanden maßgeblich in den Jahren um 1986 statt [RHW86]. Der dabei entstandene Begriff Backpropagation wird heute noch oft mit dem Gradientenabstiegs-Lernverfahren in Verbindung gebracht, wobei ersterer in Fachkreisen für die allgemeine Fehlerrückführung steht.

Wie bereits erwähnt, wird zum Erreichen eines Fehlerminimums zuerst die Änderung des Fehlers ∂E nach der Änderung der Gewichte $\partial w_{ij}^{(l)}$ benötigt. Für eine Schicht kann dieser Gradient in Form einer Jacobimatrix zusammengefasst werden:

$$J^{(l)} = \frac{\partial E}{\partial \mathbf{w}^{(l)}} = \begin{bmatrix} \frac{\partial E}{\partial w_{1,1}^{(l)}} & \cdots & \frac{\partial E}{\partial w_{1,m}^{(l)}} \\ \vdots & \ddots & \vdots \\ \frac{\partial E}{\partial w_{n,1}^{(l)}} & \cdots & \frac{\partial E}{\partial w_{n,m}^{(l)}} \end{bmatrix}. \tag{5.14}$$

Um den Fehler E nach einem Netzgewicht $w_{ij}^{(l)}$ ableitbar zu machen, muss der Gradient um den Ausgabewert eines Neurons $o_i^{(l)}$ erweitert werden:

$$\frac{\partial E}{\partial w_{ij}^{(l-1)}} = \frac{\partial E}{\partial o_i^{(l)}} \cdot \frac{\partial o_i^{(l)}}{\partial w_{ij}^{(l-1)}}. \tag{5.15}$$

Hierbei wird berücksichtigt, dass die Gewichte $w_{ij}^{(l-1)}$ einen um -1 verschobenen Index gegenüber dem, ihnen verknüpften Ausgangsneuron der Schicht (l) haben. Jetzt kann die Berechnung sukzessive für jede Schicht rückwärts durchlaufen werden.

1. Ausgangsschicht

Der Ausgang des neuronalen Netzes entspricht dem Ausgang des letzten Neurons $y = o_{i=1}^{(3)}$. Mit der Gleichung des Fehlerquadrates (5.13) kann der erste Teil der Ableitung (5.15) in Abhängigkeit vom Ausgang y aufgelöst werden:

$$\frac{\partial E}{\partial o_1^{(3)}} = \frac{\partial \frac{1}{2}(y - y_{soll})^2}{\partial y} = (y - y_{soll}). \tag{5.16}$$

Für den zweiten Teil wird noch einmal eine Erweiterung nötig. Dabei kann beachtet werden, dass für den Neuronenausgang $o_{i=1}^{(3)} = s^{(3)}(x_1^{(3)})$ der Indize „i=1“ gilt, da dieser dem einzigen Ausgang des neuronalen Netzes entspricht:

$$\frac{\partial o_1^{(3)}}{\partial w_{1,j}^{(2)}} = \frac{\partial s^{(3)}(x_1^{(3)})}{\partial w_{1,j}^{(2)}} = \frac{\partial s^{(3)}(x_1^{(3)})}{\partial x_1^{(3)}} \cdot \frac{\partial x_1^{(3)}}{\partial w_{1,j}^{(2)}}. \tag{5.17}$$

Die Ableitung der linearen Aktivierungsfunktion (5.4) in der Ausgangsschicht ergibt sich zu:

$$\frac{\partial s^{(3)}(x_1^{(3)})}{\partial x_1^{(3)}} = s'^{(3)}(x_1^{(3)}) = 1. \tag{5.18}$$

Bei der Ableitung der Eingänge der Aktivierungsfunktion nach den Gewichten, bleiben mit Gleichung (5.3) die Neuronen der vorangehenden Schicht übrig:

$$\frac{\partial x_1^{(3)}}{\partial w_{1,j}^{(2)}} = o_j^{(2)}. \tag{5.19}$$

Zusammenfassend ergibt sich also für die Ableitung des Fehlers nach den Gewichten:

$$\begin{aligned} \frac{\partial E}{\partial w_{1,j}^{(2)}} &= (y - y_{soll}) \cdot s^{(2)}(w_{1,j}^{(1)} \cdot o_j^{(2)}) \\ &= (y - y_{soll}) \cdot o_j^{(2)}. \end{aligned} \tag{5.20}$$

2. Verdeckte Schicht

Mit den gefundenen Gleichungen konnte bereits ein Zusammenhang zwischen dem Netzwerkfehler E und den Gewichten $w_{ij}^{(2)}$ der verdeckten Schicht gefunden werden. Das Vorgehen muss also für die Gewichte $w_{ij}^{(1)}$ der Eingangsschicht wiederholt werden. Um eine Ableitung des Fehlers nach den Gewichten der Eingangsschicht bilden zu können, wird diese um das Ausgangsneuron $o_1^{(3)}$ und die Neuronen der verdeckten Schicht $o_i^{(2)}$ erweitert:

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = \frac{\partial E}{\partial o_1^{(3)}} \frac{\partial o_1^{(3)}}{\partial o_i^{(2)}} \frac{\partial o_i^{(2)}}{\partial w_{ij}^{(1)}}. \tag{5.21}$$

Die Ableitung des Neuronenausgangs $o_i^{(2)}$ ist ähnlich zum bereits beschriebenen Vorgehen aus Gleichung (5.17) bis (5.19). Das Selbstbildnis der sigmoiden Aktivierungsfunktion (5.6) bietet an dieser Stelle eine deutliche Reduzierung des Rechenaufwandes, so können direkt die Ergebnisse der Feed-Forward Berechnung genutzt werden. Ohne diese Eigenschaft müsste die Ableitung direkt aus Gleichung (5.7) berechnet werden, was offensichtlich in höherem Rechenaufwand resultieren würde.

$$\frac{\partial o_i^{(2)}}{\partial w_{ij}^{(1)}} = \frac{\partial s^{(2)}(x_i^{(2)})}{\partial w_{ij}^{(1)}} = \frac{\partial s^{(2)}(x_i^{(2)})}{\partial x_i^{(2)}} \frac{\partial x_i^{(2)}}{\partial w_{ij}^{(1)}} = (1 - (o_i^{(2)})^2) \cdot o_j^{(1)}. \tag{5.22}$$

Die Ableitung der einzelnen Neuronenausgänge untereinander muss auch wieder um den Eingang der Aktivierungsfunktion erweitert werden.

$$\frac{\partial o_1^{(3)}}{\partial o_i^{(2)}} = \frac{\partial s^{(3)}(x_1^{(3)})}{\partial o_i^{(2)}} = \frac{\partial s^{(3)}(x_1^{(3)})}{\partial x_1^{(3)}} \frac{\partial x_1^{(3)}}{\partial o_i^{(2)}}. \tag{5.23}$$

Der erste Teil der zwei Terme von (5.23) ist der bereits bekannte Zusammenhang (5.18). Beim zweiten Teil bleiben aus Gleichung (5.3) die Gewichte der zweiten Schicht übrig:

$$\frac{\partial x_1^{(3)}}{\partial o_i^{(2)}} = w_{1,i}^{(2)}. \tag{5.24}$$

Die Gleichungen (5.16), (5.22) und (5.24) erbeben zusammengefasst den gesuchten Zusammenhang zwischen dem Fehler E und den Gewichten der Eingangsschicht $w_{ij}^{(1)}$:

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}^{(1)}} &= (y - y_{soll}) \cdot w_{1,i}^{(2)} \cdot s'^{(2)}(w_{ij}^{(1)} \cdot o_i^{(2)}) \cdot o_j^{(1)} \\ &= (y - y_{soll}) \cdot w_{1,i}^{(2)} \cdot (1 - (o_i^{(2)})^2) \cdot o_j^{(1)}. \end{aligned} \tag{5.25}$$

Mit den Ableitungen (5.20) und (5.25) wurde zwar inzwischen ein Zusammenhang zwischen dem Fehler E und den Gewichten gefunden, ohne eine Rückführung des Fehlers findet allerdings keine Adaption statt. Grundsätzlich wird der Lernalgorithmus iterativ zu jedem neuen Zeitpunkt t ausgeführt. Dabei werden die Netzgewichte nach folgender Regel verändert:

$$w_{ij,t+1} = w_{ij,t} + \Delta w_{ij,t}. \tag{5.26}$$

Die alten Gewichte werden also mit einem $\Delta w_{ij,t}$ verändert, wofür die bereits gefundenen Zusammenhänge nach [Roj96] verwendet werden können:

$$\Delta w_{ij}^{(l)} = \mu \cdot \frac{\partial E}{\partial w_{ij}^{(l)}}, \tag{5.27}$$

Die Lernrate μ beeinflusst dabei die Geschwindigkeit der Adaption. Ähnlich wie bei einem einfachen P-Regler in einem linearen System, kann die Lernrate bei zu kleinen Werten zu einer zu langsamen Adaption führen, wohingegen eine zu große Lernrate zu Oszillationen führt. Die Lernrate ist damit ein Faktor, der mit Vorwissen jedem neuen Lernszenario angepasst werden muss.

Zusammenfassend kann der Algorithmus wie folgt aufgestellt werden:

0.Schritt: Netzgewichte zufällig setzen

$\mathbf{w}^{(1)} = \text{rand}(-1..1)$
$\mathbf{w}^{(2)} = \text{rand}(-1..1)$

1.Schritt: Ausgang und Fehler berechnen

$$\text{Forward Propagation} : y = s^{(3)}\left(\mathbf{w}^2 \cdot \vec{s}^{\,(2)}(\mathbf{w}^{(1)} \cdot \vec{s}^{\,(1)}(\vec{\underline{x}}))\right)$$

$$E = \frac{1}{2}(y - y_{soll})^2$$

2.Schritt: Gewichtsänderung der Ausgangsschicht berechnen

$$\Delta \mathbf{w}_t = -\mu \frac{\partial E}{\partial \mathbf{w}_t}$$

4.Schritt: Neue Gewichte setzen

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \Delta \mathbf{w_t}$$

Gehe zu 1.Schritt

5.3.2 Gradientenabstiegs-Lernverfahren in Echtzeit

Um eine Abschätzung zur Anwendbarkeit im Echtzeitsystem zu bilden, muss die CPU-Last in Folge der neuronalen Netze mit Gradientenabstiegs-Lernverfahren auf dem verwendeten OMAP-3530 Prozessor betrachtet werden. Hierzu wurde die CPU-Last in Abhängigkeit von der verwendeten Neuronenzahl und der daraus resultierenden Anzahl von Verbindungsgewichten [Lan13] untersucht. Die CPU-Last kann als ungefähres Maß der Ausführungszeit des Algorithmus betrachtet werden. Das bedeutet 1% CPU-Last entspricht ca. 0.1ms Ausführungszeit bei 100Hz Ausführungstakt.

Zu beobachten ist ein Start-Offset von 7% CPU-Last, welches durch die parallele Ausführung der Flugzeugsimulation erklärt wird. Darauf folgt ein nahezu linearer Anstieg der CPU-Last bis 193 Verbindungsgewichten, wie in Grafik 5.5 dargestellt wird. Ab 193 Gewichten sinkt die CPU-Last leicht ab. Um Fehler auszuschließen, wurden an dieser Stelle die Trainings-Ergebnisse auf der embedded Hardware und der PC-Simulation verglichen und keine Abweichungen festgestellt. Wie es zu diesem Effekt kommt bleibt offen, allerdings werden weitere Untersuchungen diesbezüglich nicht als zielführend betrachtet. Der Speicherverbrauch des neuronalen Netzes, hier max. 1300kB, kann in Anbetracht der verfügbaren 500 MByte RAM vernachlässigt werden.

Zusammenfassend wird an dieser Stelle festgehalten: Die neuronalen Netze mit dem Gradientenabstiegs-Lernverfahren sind für Echtzeitanwendungen auf dem OMAP-

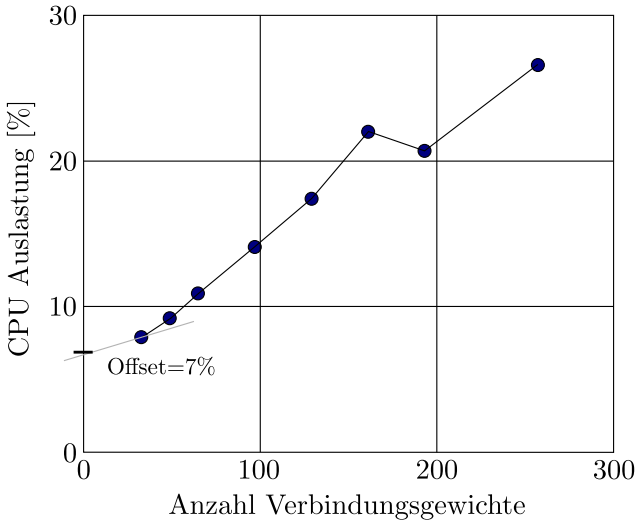


Abbildung 5.5.: CPU-Last des Gradientenabstiegs-Lernverfahrens mit 100Hz, Quelle: [Lan13].

3530 geeignet, bilden aber mit Neuronenanzahl und Anzahl der Netze einen limitieren Faktor der Möglichkeiten.

5.3.3 Variationen der Backpropagation

Wie bereits gezeigt wurde, kann mit dem Gradientenabstiegs-Lernverfahren ein adaptives Netz implementiert werden. Als Weiterentwicklung dieses Lernverfahrens wird für die anstehenden Flugversuche aus folgenden Gründen das Lernverfahren Sliding-Mode-Backpropagation (SMC-BP) untersucht: Ein wesentlicher Nachteil des Gradientenabstiegs-Lernverfahren ist die zu wählende Lernrate μ, die für jedes Lernszenario neu angepasst werden muss. Eine bekannte Weiterentwicklung dieser Lernmethode bietet das sogenannte „Levenberg-Marquardt-Lernverfahren“ [YW11]. Dieses konnte in Simulationen bereits wesentlich bessere Adaptionseigenschaften gegenüber dem Gradientenabstiegs-Lernverfahren vorweisen [Krü12]. Es soll nicht unerwähnt bleiben, dass zudem eine ganze Algorithmen-Schar an verschiedenen Lernverfahren existiert. Um einen kleinen Eindruck der möglichen Lernmethoden zu ermöglichen, wird an dieser Stelle mit Grafik 5.6 auf [Ngu06] verwiesen.

Gerade bei Verkehrsflugzeugen ist aber die Zertifizierung und ein damit verbundener Nachweis der Stabilität von Interesse. Mit diesem Hintergrund konnte in [Krü12] ein Stabilitätsnachweis der Sliding-Mode-Backpropagation in Kombination mit der

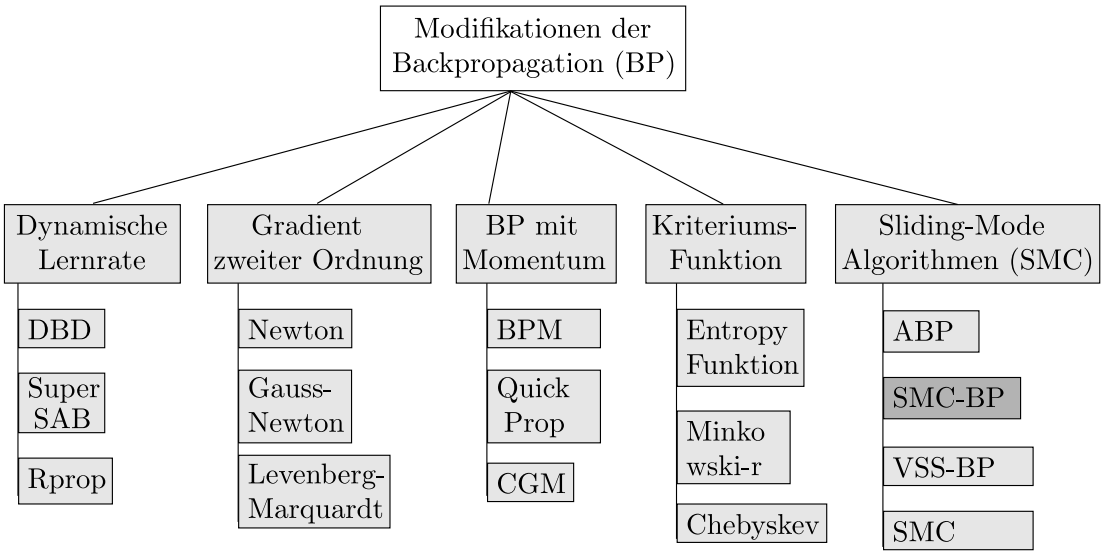


Abbildung 5.6.: Übersicht über die Variationen des Backpropagation Algorithmus, Quelle: [Ngu06].

dynamischen Inversion geführt werden. Ferner zeigte sich, dass die Sliding-Mode-Backpropagation im Rahmen der dynamischen Inversion in der Lage ist, die Lernrate μ adaptiv der Situation anzupassen.

5.3.4 Sliding-Mode-Backpropagation

Die Sliding-Mode-Backpropagation, auch Gleitzustandslernverfahren genannt, ist eine Kombination des Gradientenabstiegs-Lernverfahrens mit der, aus der Regelungstechnik bekannten Sliding-Mode-Regelung, bzw. der Gleitzustandsregelung. Damit befindet sich diese Lernmethode im Raum der Sliding-Mode-Trainingsalgorithmen [Ngu06]. Die Sliding-Mode-Regelung wiederum bildet eine Untergruppe der strukturvariablen Regelung [ES98], die hier kurz einleitend vorgestellt werden soll. Die Ursprünge der strukturvariablen Regelung wurden bereits zwischen 1940 und 1960 entwickelt, wobei ihre stabilisierenden Eigenschaften [DY10, FLT55] bewiesen wurden.

Die Grundidee der strukturvariablen Regelung ist es, einen Regler in verschiedene Teilregler zu unterteilen, wie in Grafik 5.7 abgebildet. In Abhängigkeit von Schaltfunktionen $\vec{S}(\vec{x})$, hier als Schaltlogik dargestellt, wird dann zwischen den verschiedenen Teilreglern hin und her geschaltet, je nach dem, in welchem Zustand sich das zu regelnde System befindet. So wird sichergestellt, dass für jeden Systemzustand ein stabilisierender Teilregler agiert.

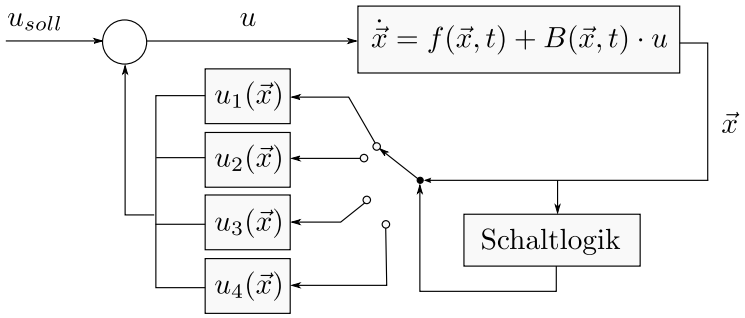


Abbildung 5.7.: Beispiel einer strukturvariablen Regelung.

Nach den ersten Formulierungen der strukturvariablen Regelung wurden in den 1950er-Jahren die ersten Sliding-Modes vorgestellt und die Entwicklung der Sliding-Mode-Regelung, auch Schaltregelung genannt, maßgeblich in der ehemaligen Sowjetunion untersucht und weiterentwickelt [XJXU02, Eme69, Utk78]. Nach diesem kurzen geschichtlichen Exkurs zu den Ursprüngen des verwendeten Lernverfahrens, wird im Folgenden die Grundidee der Sliding-Mode-Regelung beschrieben, die einen zentralen Baustein des Gleitzustandslernverfahrens bildet. Eine einfache Einführung in dieses Thema bietet zudem [Heb95], eine detaillierte findet sich in [Ngu06, Kha02].

Gegeben sei das folgende zu regelnde System:

$$\dot{\vec{x}} = \vec{f}(\vec{x}) + B(\vec{x}) \cdot \vec{u}, \tag{5.28}$$

mit dem Zustandsvektor $\vec{x} = [x_1, x_2, ..., x_n]^T$, der nichtlinearen Eingangsmatrix $B(\vec{x}, t)$ des Eingangsvektors $\vec{u} = [u_1, u_2, ..., u_n]^T$ sowie der nichtlinearen Systemfunktion $\vec{f}(\vec{x}, t)$. Die Reglergleichung $\vec{u}(\vec{x}, t)$ wird nun in Abhängigkeit von den bereits erwähnten Schaltfunktionen $\vec{S}(\vec{x}) = [S_1(\vec{x}), S_2(\vec{x},), ..., S_n(\vec{x})]^T$ nach folgendem Stellgesetz hin und her geschaltet:

$$\vec{u}(\vec{x}, t) = \begin{cases} \vec{u}^+(\vec{x}, t), & \text{für} \quad \vec{S}(\vec{x}) > 0 \\ \vec{u}^-(\vec{x}, t), & \text{für} \quad \vec{S}(\vec{x}) < 0 \end{cases}. \tag{5.29}$$

Für die Auslegung des Sliding-Mode-Reglers existieren verschiedene Ansätze [Ngu06, Krü12], die mit folgenden Sätzen zusammengefasst werden können: Der Regler wird so ausgelegt, dass die Systemzustände zur sogenannten Schaltlinie (bei höheren Ordnungen auch Schaltflächen, Hyperflächen, bzw. Schaltmannigfaltigkeit) konvergieren. Dies bedeutet, dass die Existenz und Erreichbarkeit der Schaltlinie vorausgesetzt wird. Das Prinzip soll anhand von Grafik 5.8 veranschaulicht werden. Ist die Schaltlinie $S = 0$ erreicht, wird diese zur Gleitlinie (engl. sliding mode) und die

Systemzustände gleiten im Idealfall entlang dieser Linie, wodurch der Gleitzustand erreicht ist. Der Regler $\vec{u}$ ist dabei so auszulegen, dass das System asymptotisch stabilisiert wird.

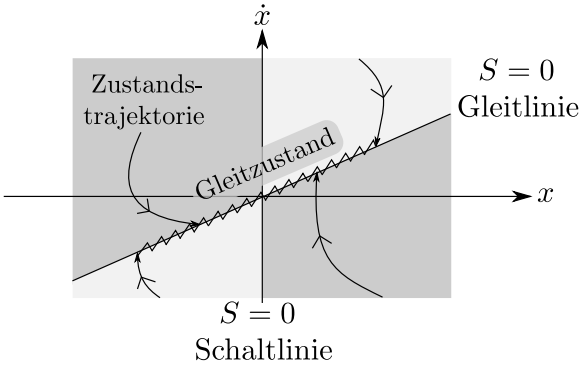


Abbildung 5.8.: Beispiel der Schaltlinie $S(\vec{x}) = \dot{x} - 0,3x$.

Die Schaltfunktion wird im Folgenden, angelehnt an [Utk92, Krü12], definiert zu:

$$S = \sum_{n}^{i=1} \lambda_i \cdot x^{(i-1)} = 0, \text{ mit } \lambda_n = 1. \tag{5.30}$$

5.3.5 Lernen mit Sliding-Mode-Backpropagation

Die erste Formulierung strukturvariabler Regelung in Kombination mit neuronalen Netzen wird in in [PM98] erwähnt. Diese Technologie wurde weiter untersucht, wobei [KEE01, YK09] und [EK02] eine gute Allgemeinübersicht bieten. Allen Verfahren ist die Betrachtung von Netzwerkgrößen als Zustandsvektor $\vec{x}$ in der Schaltfunktion $S(\vec{x})$ gemein. Hauptsächlich wird entweder der quadratische Fehler E oder der Ausgabefehler ϵ als Zustandsgröße für die Schaltfunktion definiert. Weiterhin können die Gradientenmatrix $\Delta E(\vec{w})$ [PMBB98] oder die Gewichte $\Delta\vec{w}$ selbst [TK01] herangezogen werden.

Im Folgenden wird der Ausgabefehler ϵ als Zustandsgröße für die Schaltfunktion verwendet. Dies hat wesentliche Vorteile: Nach [Krü12] wird ein Paradigmenwechsel der Betrachtungsweise von neuronalen Netzen möglich: der Betrachtung des Netzes als zu regelndes System, wie in Bild 5.9 dargestellt. Mit dem Ausgabefehler ϵ als Zustandsgröße kann so, aus regelungstechnischer Sicht, eine Stabilitätsuntersuchung auf derart trainierte neuronale Netze angewendet werden.

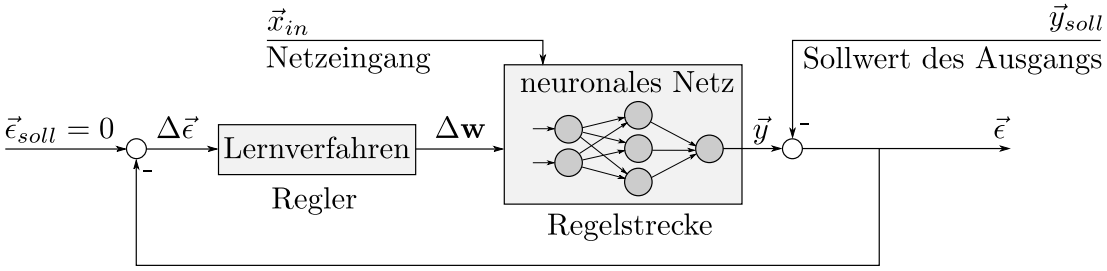


Abbildung 5.9.: Betrachtung des neuronalen Netzes als Regelkreis.

Hinzu kommt, dass die Verwendung von ΔE bzw. Δw aus entwurfstechnischer Sicht wesentlich unvorteilhafter ist, da die Reglerauslegung für jedes Element des Netzwerkes einzeln betrachtet werden müsste. Als letztes Argument zeigt [Sch11a] die überlegene Performance gegenüber anderen Trainingsverfahren.

Mit der Einführung von ϵ als Zustandsgröße ergibt sich aus Gleichung (5.30) folgende Schaltlinie S:

$$S = \dot{\epsilon} + \lambda\epsilon. \tag{5.31}$$

Um die Erreichbarkeit und und Existenz des Gleitzustandes zu garantieren (vgl. [Krü12]), wird der Ansatz von Ljapunow, basierend auf [SIK87], mit der Schaltfunktion $S(\epsilon)$ verknüpft.

$$|S_{t+1}| \overset{!}{<} |S_t|. \tag{5.32}$$

In der klassischen Rückpropagation wird die Lernrate μ, wie bereits erwähnt, empirisch für jedes Lernszenario neu festgelegt, was die regelungstechnische Anwendung auf eine Anwendungsbandbreite einschränkt. Dieser Nachteil kann durch die Anwendung der Schaltlinie S_t auf das Netz-Training umgangen werden, wie im Folgenden angelehnt an [Krü12] gezeigt wird. Dazu wird die Lernregel der Rückpropagation (5.27) aus dem Gradientenabstiegs-Lernverfahren variiert:

$$\Delta\mathbf{w} = \left(\frac{\partial\vec{y}(\mathbf{w}, x)}{\partial\mathbf{w}}\right)^T \mu \cdot diag(sign(S_t)) \cdot |\epsilon|. \tag{5.33}$$

Zur Bestimmung der Lernrate μ muss nun die Gleichung (5.32) einer genaueren Betrachtung unterzogen werden. Mit der zeitlichen Approximation des Fehlers ϵ für kleine Zeiten T_s zu

$$\dot{\epsilon}(t) \approx \frac{1}{T_s}(\epsilon_t - \epsilon_{t-1}), \tag{5.34}$$

wobei der Fehler zum Zeitpunkt t auch darstellbar ist als

$$\epsilon(t) = \epsilon_t \tag{5.35}$$

sowie

$$\epsilon(t - T_s) = \epsilon_{t-1}, \tag{5.36}$$

kann damit die Schaltlinie S_t bzw. S_{t+1} folgendermaßen dargestellt werden:

$$\begin{aligned} S_t &= \dot{\epsilon}_t + \lambda\epsilon_t = (\lambda + \frac{1}{T_S})\epsilon_t - \frac{1}{T_S}\epsilon_{t-1} \\ S_{t+1} &= \dot{\epsilon}_{t+1} + \lambda\epsilon_{t+1} = (\lambda + \frac{1}{T_S})\epsilon_{t+1} - \frac{1}{T_S}\epsilon_t. \end{aligned} \tag{5.37}$$

Da der Fehler ϵ_{t+1} nicht bekannt ist, wird dieser nach Gleichung

$$\epsilon_{t+1} = \epsilon_t + \Delta\epsilon_t \tag{5.38}$$

berechnet. Die Änderung des Fehlers $\Delta\epsilon_t$ setzt sich aus dem gewünschten Ausgang $y_{s,t}$ sowie dem tatsächlichen Ausgang y_t des Netzes

$$\Delta\epsilon_t = \Delta y_{s,t} - \Delta y_t = (y_{s,t+1} - y_{s,t}) - (y_{t+1} - y_t) \tag{5.39}$$

zusammen. Mit dem Ziel, geeignete Werte für die Lernrate μ in der Schaltlinie S_{t+1} zu finden, wird mit den Gleichungen (5.37) und (5.38) die Ungleichung (5.32) weiter ausformuliert zu

$$|S_t| \stackrel{!}{>} \left| \left(\lambda + \frac{1}{T_s}\right)(\epsilon_t + \Delta\epsilon_t) - \frac{1}{T_s}\epsilon_t \right|. \tag{5.40}$$

Ähnlich dem Gradientenabstiegs-Lernverfahren kann zur weiteren Berechnung die Ableitung des Ausgangs y nach den Gewichten $\mathbf{w}$ betrachtet werden. Mit der Annäherung durch eine Taylorreihe inkl. der partiellen Ableitungen

$$\Delta\vec{y}_t = \frac{\partial y_t(\mathbf{w}, \vec{x})}{\partial \mathbf{w}_t}\Delta\mathbf{w}_t + \frac{\partial y_t(\mathbf{w}, \vec{x})}{\partial \vec{x}_t}\Delta\vec{x}_t, \tag{5.41}$$

wird hierfür wieder eine Jacobi-Matrix erstellt:

$$J_{y,t}^{(l)} = \frac{\partial y_t(\mathbf{w}_t, \vec{x}_t)}{\partial \mathbf{w}_t^{(l)}} = \begin{bmatrix} \frac{\partial y_t}{\partial w_{1,1,t}^{(l)}} & \cdots & \frac{\partial y_t}{\partial w_{1,m,t}^{(l)}} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_t}{\partial w_{n,1,t}^{(l)}} & \cdots & \frac{\partial y_t}{\partial w_{n,m,t}^{(l)}} \end{bmatrix}. \tag{5.42}$$

Die einzelnen Ableitungen können analog zu den Gleichungen (5.23) bis (5.24) mit dem Zusammenhang $y = o_{j=1}^{(3)}$ bestimmt werden, wie folgend dargestellt:

$$\begin{aligned} \frac{\partial y(\mathbf{w}, \vec{x})}{\partial w^{(1)}} &= \frac{\partial o_{j=1}^{(3)}}{\partial o_j^{(2)}} \cdot \frac{\partial o_j^{(2)}}{\partial w^{(1)}} \\ \frac{\partial y(\mathbf{w}, \vec{x})}{\partial w^{(2)}} &= \frac{\partial o_{j=1}^{(3)}}{\partial w^{(2)}}. \end{aligned} \tag{5.43}$$

Mit diesen Zusammenhängen kann Gleichung (5.40) weiter ausformuliert werden,

$$|S_t| \overset{!}{>} \left| \left(\lambda + \frac{1}{T_s}\right) \left(\epsilon_t + \Delta y_{s,t} - J_{y,t} \Delta \mathbf{w}_t - \frac{\partial y_t(\mathbf{w}_t, \vec{x}_t)}{\partial \vec{x}_t} \Delta \vec{x}_t\right) - \frac{1}{T_s} \epsilon_t \right|. \tag{5.44}$$

Mit dem Einsetzen der neuen Lernregel (5.33) ist die Gleichung (5.44) endgültig von der Lernrate μ abhängig.

$$|S_t| \overset{!}{>} \left| \left(\lambda + \frac{1}{T_s}\right) \left(\epsilon_t + \Delta y_{s,t} - J_{y,t} \cdot J_{y,t}^T \cdot \mu \cdot \operatorname{diag}(\operatorname{sign}(S_t)) \cdot |\epsilon| - \frac{\partial y_t}{\partial \vec{x}_t} \Delta \vec{x}_t\right) - \frac{1}{T_s} \epsilon_t \right|. \tag{5.45}$$

Der Übersicht halber wird nun diese Formulierung mit den Parametern a und b folgend substituiert:

$$\begin{aligned} a &= (\lambda + \frac{1}{T_s})(\epsilon_t + \Delta y_{s,t} - \frac{\partial y_t}{\partial \vec{x}_t} \Delta \vec{x}_t) - \frac{1}{T_s} \epsilon_t \\ b &= (\lambda + \frac{1}{T_s}) \cdot \left(J_{y,t} \cdot J_{y,t}^T \cdot \mu \cdot \operatorname{diag}(\operatorname{sign}(S_t)) \cdot |\epsilon|\right)_t, \end{aligned} \tag{5.46}$$

womit sich Gleichung 5.45 vereinfacht zu:

$$|S_t| \overset{!}{>} |a - \mu b|. \tag{5.47}$$

Mit dieser Ungleichung werden durch eine Fallanalyse Grenzen für die Lernrate μ bestimmt.

Folgende Fälle können auftreten:

Für $S_t > 0$ gilt:

$$S_t \overset{!}{>} a - \mu \cdot b \qquad \text{für } a - \mu \cdot b > 0 \tag{5.48}$$

oder

$$S_t \overset{!}{<} a - \mu \cdot b \qquad \text{für } a - \mu \cdot b < 0 \tag{5.49}$$

und für $S_t < 0$ gilt:

$$-S_t \overset{!}{>} a - \mu \cdot b \qquad \text{für } a - \mu \cdot b > 0 \tag{5.50}$$

oder

$$-S_t \overset{!}{<} a - \mu \cdot b \qquad \text{für } a - \mu \cdot b < 0. \tag{5.51}$$

Zusammenfassend ergeben die Bedingungen für die Lernrate μ in Abhängigkeit des Parametervorzeichens von b:

$$\begin{aligned}
&\{\mu > \frac{-S+a}{b} \wedge \mu < \frac{a}{b}\} \vee \{\mu < \frac{S+a}{b} \wedge \mu > \frac{a}{b}\} && \text{für } b > 0 \\
&\{\mu < \frac{-S+a}{b} \wedge \mu > \frac{a}{b}\} \vee \{\mu > \frac{S+a}{b} \wedge \mu < \frac{a}{b}\} && \text{für } b < 0 \\
&\{\mu > \frac{S+a}{b} \wedge \mu < \frac{a}{b}\} \vee \{\mu < \frac{-S+a}{b} \wedge \mu > \frac{a}{b}\} && \text{für } b > 0 \\
&\{\mu < \frac{S+a}{b} \wedge \mu > \frac{a}{b}\} \vee \{\mu > \frac{-S+a}{b} \wedge \mu < \frac{a}{b}\} && \text{für } b < 0.
\end{aligned} \tag{5.52}$$

Wählt man nun $\frac{-S+a}{b} = \frac{-S}{b} + \frac{a}{b}$ bzw. $\frac{S+a}{b} = \frac{S}{b} + \frac{a}{b}$ als eigentliche Grenzen, so ist ersichtlich, dass nach Auswahl einer Lernrate μ innerhalb dieser Grenzen immer nur eine Bedingung $\mu > \frac{-S+a}{b} \wedge \mu < \frac{a}{b}$ oder $\mu < \frac{S+a}{b} \wedge \mu > \frac{a}{b}$ (analog für alle Fälle) eine Lösung ergeben kann. Damit kann die Ungleichung (5.52) auch wie folgt vereinfacht werden:

$$\begin{aligned}
&\frac{-S+a}{b} < \mu < \frac{S+a}{b} && \text{für } (S > 0 \vee b > 0) \wedge (S < 0 \vee b < 0) \\
&\frac{S+a}{b} < \mu < \frac{-S+a}{b} && \text{für } (S > 0 \vee b < 0) \wedge (S < 0 \vee b > 0).
\end{aligned} \tag{5.53}$$

Damit ist der Gleitzustand und die Stabilität für jedes μ innerhalb dieser Grenzen garantiert. Beachtet man aber dabei, dass für die Herleitung der Grenzen verein-

fachende Annahmen getroffen wurden, wird häufig der Mittelwert für die Lernrate μ herangezogen. Damit sind genügend Stabilitätsreserven innerhalb der oberen und unteren Grenze vorhanden [Krü12, Lan13].

5.3.6 Sliding-Mode-Backpropagation in Echtzeit

Analog zum Gradientenabstiegs-Lernverfahren (vgl. Unterkapitel 5.3.2) wird zur Abschätzung der Anwendbarkeit im Echtzeitsystem die CPU-Last der Sliding-Mode-Backpropagation untersucht, wie in Grafik 5.10 dargestellt. Die Sliding-Mode-Backpropagation wird dazu mit 100Hz Wiederholrate auf dem OMAP-3530 ausgeführt. Beobachtbar ist ein fast identischer Ressourcenverbrauch im Vergleich zum Gradientenabstiegs-Lernverfahren.

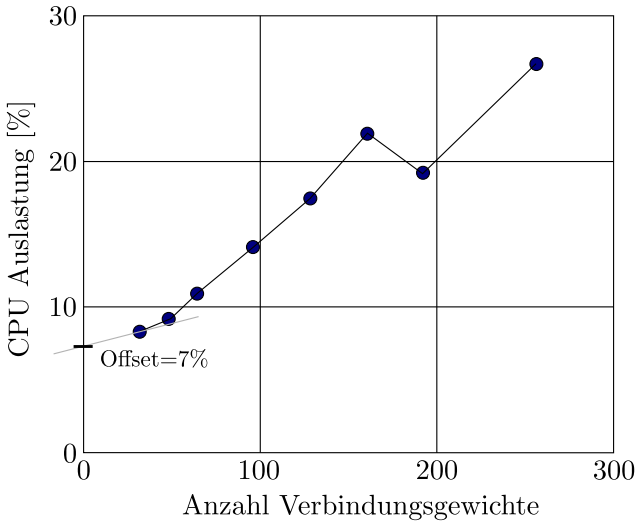


Abbildung 5.10.: CPU-Last der Sliding-Mode-Backpropagation mit 100Hz auf dem OMAP-3530 Prozessor, Quelle: [Lan13].

Zusammenfassend wird an dieser Stelle festgehalten: Die neuronalen Netze mit der Sliding-Mode-Backpropagation sind für Echtzeitanwendungen auf dem OMAP-3530 geeignet, bilden aber, wie beim Gradientenabstiegs-Lernverfahren, mit Neuronenanzahl und Anzahl der Netze einen limitieren Faktor der Möglichkeiten. Dieses Ergebnis ist insoweit von Bedeutung, da bereits in [Krü12, Lan13, Pla10, Möß09] die deutliche Überlegenheit des Sliding-Mode-Trainings gegenüber der klassischen Backpropagation gezeigt werden konnte.

5.4 Erweiterung der dynamischen Inversion um neuronale Netze

Wie bereits gezeigt wurde, können Linearregler oder die dynamische Inversion mit erheblichen Modellfehlern oder nicht modellierten Fehlergrößen an ihre Grenzen stoßen. Für die Kompensation bieten sich daher besonders neuronale Netze an. Um die Darstellung der Reglerstrukturerweiterung mit diesen adaptiven Elementen zu vereinfachen, soll im Folgenden wieder ein SISO-System verwendet werden. Da, wie bereits erwähnt, neuronale Netze als universelle Approximatoren betrachtet werden, kann diese Eigenschaft nach [SH89, Krü12] folgend dargestellt werden.

$$f_{NN} : \underline{X} \in \mathbb{R}^n \rightarrow Y \in \mathbb{R}^m. \tag{5.54}$$

Neuronale Netze bilden mit der Forward-Propagation also eine Abbildung der Untermenge $\{\underline{X}\}$ in eine Untermenge $\{Y\}$ ab. Ferner gilt für die nichtlineare, differenzierbare Funktion $f(\vec{\underline{x}}) : \mathbb{R}^n \rightarrow \mathbb{R}^m$ in einer beschränkten und abgeschlossenen Teilmenge $D \subset \mathbb{R}^n$:

$$f(\vec{\underline{x}}) = s^{(3)}\left(\mathbf{w}_*^{(2)} \cdot \vec{s}^{\,(2)}(\mathbf{w}_*^{(1)} \cdot \vec{\underline{x}})\right) + \epsilon_A(\vec{\underline{x}}) \quad \forall \vec{\underline{x}} \in D \subset \mathbb{R}^n. \tag{5.55}$$

Dies bedeutet, dass die Forward-Propagation mit den optimalen Gewichten $\mathbf{w}_*^{(l)}$ die nichtlineare Funktion $f(\vec{\underline{x}})$ bis auf den Abbildungsfehler ϵ_A genau nachbildet. Dieser muss mit einer oberen Schranke $\overline{\epsilon}_A$ begrenzt sein, so dass gilt $||\epsilon_A|| < \overline{\epsilon}_A$.

Das Ziel der neuronalen Netze in Verbindung mit der dynamischen Inversion ist demnach die Adaption einer nichtlinearen Fehlerfunktion $\Delta(\vec{\underline{x}})$, die dem Eingang der Fehlerdynamik aus Gleichung (4.67) entspricht [LYL96].

$$\Delta(\vec{\underline{x}}) = \mathbf{w}_*^{(2)} \cdot \vec{f}^{(2)}(\mathbf{w}_*^{(1)} \cdot \vec{\underline{x}}) + \epsilon_A(\vec{\underline{x}}). \tag{5.56}$$

Im Folgenden muss also der Eingang der neuronalen Netze sowie die für das Training der neuronalen Netze benötigte Trainingsgröße definiert werden. Mit der Variable $\zeta = \vec{\chi}^T \cdot \mathbf{A}_E \cdot \mathbf{b}_E$ wird die sogenannte gefilterte Fehlergröße in Gleichung 5.57 eingeführt, wobei $\mathbf{b}_E$ (im MIMO-Fall eine Matrix) dem Eingangsvektor der Fehlerdynamik aus Gleichung (4.67) (vgl. auch Gl. (4.51)) entspricht:

$$\zeta = \vec{\chi}^T \cdot \mathbf{P}_E \cdot \mathbf{b}_E = \left[e, \dot{e}, \ldots, e^{(r-1)}\right] \underbrace{\begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1r} \\ p_{21} & p_{22} & \cdots & p_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ p_{r1} & p_{r2} & \cdots & p_{rr} \end{bmatrix}}_{\mathbf{P}_E} \underbrace{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}}_{\mathbf{b}_E}. \tag{5.57}$$

Die positiv definite Matrix $\mathbf{P}_E$ sowie $\mathbf{b}_E$ sind Teil der Ljapunow-Funktion, deren Herleitung und Zusammenhang auf der nächsten Seite vorgestellt wird. Die gefilterte Fehlergröße ζ (im MIMO-Fall ein Vektor) ist somit eine Abbildung des Fehlervektors $\vec{\chi}$ in den für das Netztraining benötigten Netzfehler ϵ aus Gleichung (5.33). Der Ausgang des neuronalen Netzes wird mit

$$\nu_{ad} = \mathbf{w}_*^{(2)} \cdot \vec{f}^{(2)}(\mathbf{w}_*^{(1)} \cdot \underline{\vec{x}}) \tag{5.58}$$

angegeben. Weiterhin wird nach [Krü12] der sogenannte robustifizierende Term ν_r für den Stabilitätsnachweis benötigt:

$$\nu_r = [k_{r0} + k_{r1} \cdot (||\mathbf{w}||_F + \bar{\mathbf{w}}_*)] \cdot \zeta, \tag{5.59}$$

wobei $||\mathbf{w}||_F$ der Frobeniusnorm der Gewichtsmatrix $\mathbf{w}$ und $\bar{\mathbf{w}}_*$ der oberen Schranke der optimalen Gewichtsmatrix entspricht, sowie k_{r0} und k_{r1} stabilitätsrelevanten Designparametern entsprechen. Mit dem robustifizierenden Term ν_r und dem Netzausgang ν_{ad} sowie der alten Pseudosteuergröße ν aus Gleichung (4.58) wird die neue Pseudosteuergröße gebildet:

$$\nu = y_{ref}^{(r)} - \vec{c}^{\,T} \cdot \vec{\chi} + \nu_{ad} + \nu_r. \tag{5.60}$$

Zum Überblick wird an dieser Stelle die entstehende Konfiguration des Referenzmodells mit PCH und Fehlerregler mit der Erweiterung um das neuronale Netz in Abbildung 5.11 dargestellt.

Der Referenzausgang bildet sich analog zu Gleichung (4.65) zu

$$y^{(r)} = y_{m,ref}^{(r)} - c^T \cdot \vec{\chi} + \nu_{ad} + \nu_r + \Delta, \tag{5.61}$$

in Abhängigkeit vom Referenz-Messausgang $y_{m,ref}^{(r)}$, robustifizierendem Term, Netzausgang, dem Regelfehler und dessen Ableitungen χ und dem Inversionsfehler Δ.

Dies wird auch durch den Vergleich der Abbildungen 4.11, 4.12 und 5.11 deutlich sichtbar.

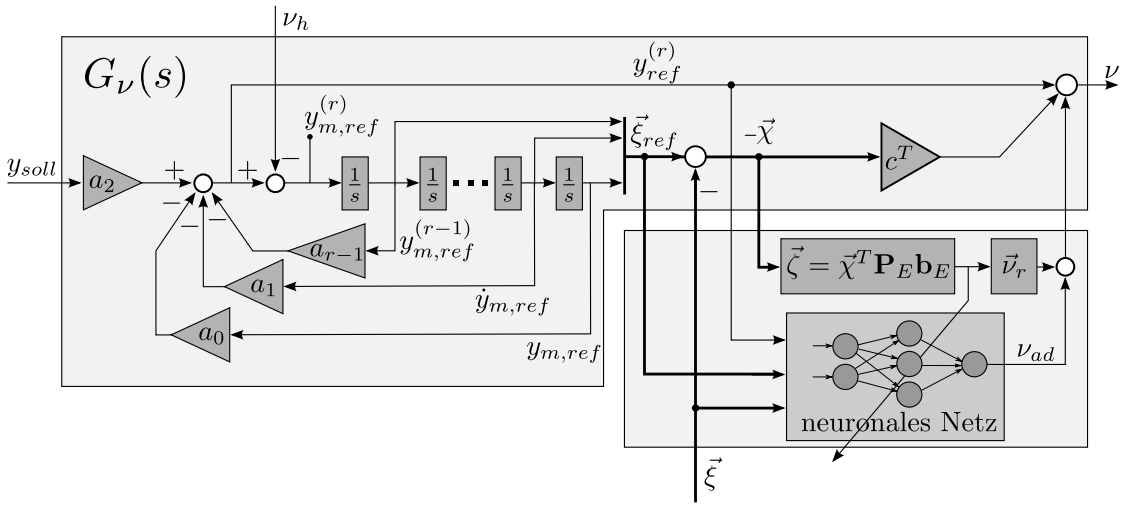


Abbildung 5.11.: Erweiterung des Referenzmodells mit Fehlerregler und PCH um neuronale Netze.

Analog zum Vorgehen zu Gleichung (4.67) wird die neue Fehlerdynamik bestimmt:

$$\dot{\vec{\chi}} = \mathbf{A}_E \vec{\chi} + \mathbf{b}_E \cdot (\Delta - \nu_{ad} - \nu_r). \tag{5.62}$$

Die bereits erwähnten Matrizen $\mathbf{A}_E$ sowie $\mathbf{P}_E$ bilden nach [LL67], angelehnt an [Krü12], die Ljapunow-Gleichung:

$$\mathbf{A}_E^T \cdot \mathbf{P}_E + \mathbf{P}_E \cdot \mathbf{A}_E = -\mathbf{Q}_E, \tag{5.63}$$

wobei $\mathbf{A}_E$ die Dynamikmatrix der Fehlerdynamik aus Gleichung (5.62) repräsentiert. Mit der frei definierbaren Matrix $\mathbf{Q}_E$, üblicherweise eine Einheitsmatrix, kann die Matrix $\mathbf{P}_E$ gewonnen werden. Hierfür wird die Ljapunow-Funktion herangezogen:

$$V(\vec{\chi}) = \vec{\chi}^T \cdot \mathbf{P}_E \cdot \vec{\chi}. \tag{5.64}$$

Als eine Voraussetzung zur Erfüllung der Stabilitätsbedingungen nach [Krü12, Hol04], wird die negative Definitheit der Ableitung von $V(\vec{\chi})$ gefordert:

$$\dot{V}(\vec{\chi}) = \dot{\vec{\chi}}^T \cdot \mathbf{P}_E \cdot \vec{\chi} + \vec{\chi}^T \cdot \mathbf{P}_E \cdot \dot{\vec{\chi}} \overset{!}{<} 0. \tag{5.65}$$

Nun kann die Definition der Fehlerdynamik aus Gleichung (5.62) auf die Ableitung der Ljapunow-Funktion übertragen werden. Der Übersichtlichkeit halber wird die Substitution $u = (\Delta - \nu_{ad} - \nu_r)$ eingeführt.

$$\dot{V}(\vec{\chi}) = [\mathbf{P}_E \cdot \vec{\chi} + \mathbf{b}_E \cdot \vec{u}] \cdot \mathbf{P}_E \cdot \vec{\chi} + \vec{\chi}^T \cdot \mathbf{P}_E \cdot [\mathbf{P}_E \cdot \vec{\chi} + \mathbf{b}_E \cdot u] \overset{!}{<} 0. \tag{5.66}$$

Durch Ausmultiplizieren und Einsetzen von Gleichung (5.63) kann letztendlich folgender Zusammenhang gefunden werden:

$$\dot{V}(\vec{\chi}) = -\vec{\chi}^T \cdot \mathbf{Q}_E \cdot \vec{\chi} + 2 \cdot \vec{\chi}^T \cdot \mathbf{P}_E \cdot \mathbf{b}_E \cdot u \overset{!}{<} 0. \tag{5.67}$$

Mit der Festlegung einer oberen Schranke der Fehlerzustandsgröße $\vec{\chi}$ wird die Abbildungsmatrix $\mathbf{P}_E$ bestimmt. Der Bestimmung der Schranke wurde in [Hol04] in ausreichendem Maße Rechnung getragen, deshalb kann an dieser Stelle davon Abstand genommen werden.

Letztendlich müssen noch die Eingangsgrößen $\underline{\vec{x}}$ der neuronalen Netze definiert werden. Hierfür haben sich die Zustandsgrößen $\vec{x}(t)$, das Referenzsignal $y_{ref}^{(r)}(t)$ und die Ableitungen des Referenzsignals $\vec{\xi}_{ref}(t)$ sowie deren zeitliche Verzögerungen, wie von [Krü12] vorgeschlagen, bewährt [Rob13]:

$$\underline{\vec{x}} = \begin{bmatrix} \vec{x}(t) \\ \vec{x}(t-1) \\ y_{ref}^{(r)}(t) \\ y_{ref}^{(r)}(t-1) \\ \vec{\xi}_{ref}(t) \\ \vec{\xi}_{ref}(t-1) \end{bmatrix}. \tag{5.68}$$

Die Verwendung der zeitlichen Verzögerungen kann damit begründet werden, dass die zu erlernende Fehlerdynamik aus einem zeitabhängigen Prozess besteht.

5.5 Umsetzung der dynamischen Inversion mit neuronalen Netzen

Mit der allgemeinen Definition der Verkopplung von neuronalen Netzen mit der dynamischen Inversion im letzten Unterkapitel, folgt nun die Implementierungsstrategie für das unbemannte Flugzeug T200, bzw. Twinstar.

Dazu wird die dynamische Inversion um neuronale Netze in der Rotationsdynamik ergänzt, wie in Abbildung 5.12 dargestellt. Die Lagedynamik bleibt von dieser Ergänzung unbeeinflusst, so dass weiterhin nur die Rotationsdynamik noch einmal genauerer Betrachtung bedarf. Eine Erweiterung der Lagedynamik um neuronale Netze (gestrichelte Linie in Abb. 5.12) ist möglich und für zukünftige Forschungsansätze vorgesehen [Krü12]. In den folgenden Betrachtungen kommt das PCH zum Einsatz. Eine Verwendung von RCH ist auch möglich, führt aber nach ersten Erkenntnissen zu keiner signifikanten Verbesserung. Wie bereits zu Grafik 4.14 erläutert, wird auf eine Unterscheidung zwischen Messung und realer Größe verzichtet.

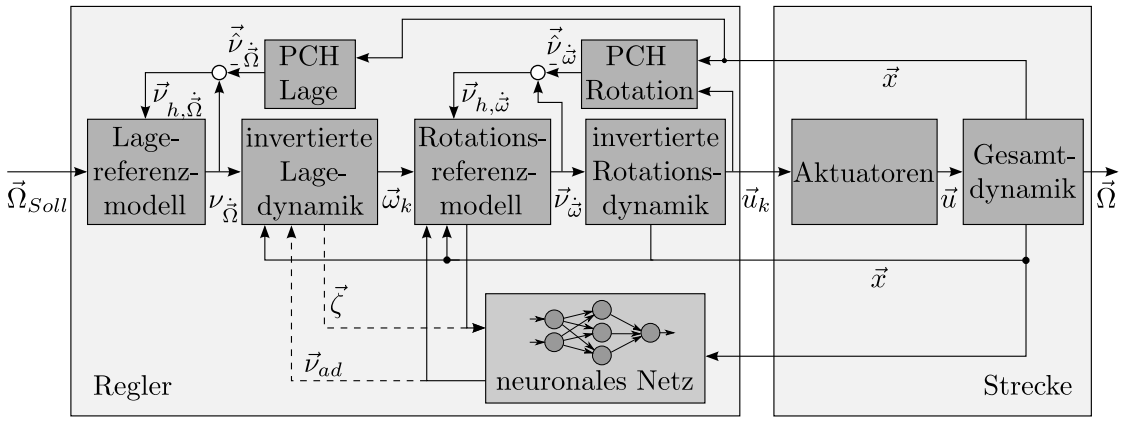


Abbildung 5.12.: Übersicht der Eulerinversion mit relativem Grad eins.

Abweichend zu Abbildung 5.12 ist die tatsächliche Implementierung der Rotationsregelung in Simulink dahingehend, dass diese nicht in Vektorform, sondern für jede der drei Achsen einzeln erfolgt, wobei aber die verwendeten Gleichungen und Darstellungen ihre Gültigkeit beibehalten.

Beginnend mit der Pseudosteuergröße, ergibt sich diese für die Rotationsdynamik aus Gleichung (5.60), vgl. auch Grafik 5.13:

$$\vec{\nu}_{\dot{\vec{\omega}}} = \tilde{\mathbf{T}}_{\vec{\omega}} \cdot (\vec{\omega}_{soll} - \vec{\omega}_{m,ref}) + \mathbf{K}_{\vec{\omega}} \cdot (\vec{\omega}_{m,ref} - \vec{\omega}) + \vec{\nu}_r + \vec{\nu}_{ad}. \tag{5.69}$$

Die Fehlergrößen setzen sich aus den Referenzsignalen und den kalibrierten Messsignalen des Drehratensensors zusammen und bilden den Fehlergrößenvektor:

$$\vec{e}_{\vec{\omega}} = \vec{\omega} - \vec{\omega}_{m,ref} = \begin{bmatrix} e_p \\ e_q \\ e_r \end{bmatrix} = \begin{bmatrix} p - p_{m,ref} \\ q - q_{m,ref} \\ r - r_{m,ref} \end{bmatrix}. \tag{5.70}$$

Da sich bei der Rotationsdynamik ein relativer Grad $r = 1$ ergeben hat, entspricht der Fehlergrößenvektor dem Fehlervektor $\vec{\chi} = \vec{e}_{\vec{\omega}}$, der keine Ableitungen beinhaltet. Weiterhin gilt für die Ableitung des Fehlers, bezüglich dem im Theorieabschnitt besprochenen Messpunkt,

$$\dot{\vec{e}}_{\vec{\omega}} = \dot{\vec{\omega}} - \dot{\vec{\omega}}_{m,ref}, \tag{5.71}$$

womit noch einmal gezeigt wird, dass die Aktuatordynamik berücksichtigt wird und somit keinen Einfluss auf die neuronalen Netze hat.

Die Fehlerdynamik kann gemäß Gleichung (5.62) mit $\mathbf{A}_{E,\vec{\omega}} = -\mathbf{K}_{\vec{\omega}}$ und der Einheitsmatrix $\mathbf{b}_{E,\vec{\omega}}$ zu

$$\dot{\vec{e}}_{\vec{\omega}} = -\mathbf{K}_{\vec{\omega}} \cdot \vec{e}_{\vec{\omega}} + \vec{\Delta}_{\omega} - \vec{\nu}_{ad,\omega} - \vec{\nu}_{r,\omega} \tag{5.72}$$

bestimmt werden. An dieser Stelle sei angemerkt, dass die Differenzialgleichungen der einzelnen Achsen entkoppelt sind, wohingegen in der Realität durch leichte Asymmetrien noch Kopplungseffekte zu erwarten sind.

Die gefilterte Fehlergröße

$$\vec{\zeta}_{\vec{\omega}} = \vec{e}_{\vec{\omega}}^{\,T} \cdot \mathbf{P}_{E,\vec{\omega}} \cdot \mathbf{b}_{E,\vec{\omega}}, \tag{5.73}$$

wird analog zu Gleichung (5.57) gebildet. Dabei entsprechen die Elemente $[k_p, k_q, k_r]$ von Matrix

$$\mathbf{P}_{E,\vec{\omega}} = \begin{bmatrix} 1/k_p & 0 & 0 \\ 0 & 1/k_q & 0 \\ 0 & 0 & 1/k_r \end{bmatrix}, \tag{5.74}$$

der Matrix $\mathbf{K}_{\vec{\omega}}$ der Fehlerdynamik.

Zusammenfassend wird das neue Referenzmodell der Rotationsdynamik in Abbildung 5.13 dargestellt, wobei die Struktur des Rotationsreglers aus Abbildung 4.16

weiterhin gilt. Offensichtlich fehlt noch die genaue Definition der Eingangsgrößen $\underline{\vec{x}}$ der neuronalen Netze, die nach dem Vorbild aus Gleichung 5.68 bestimmt werden.

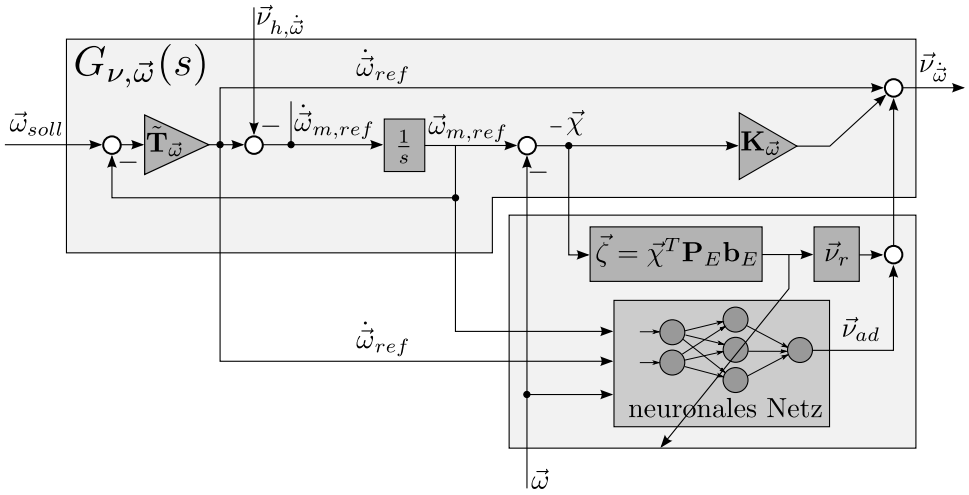


Abbildung 5.13.: Erweiterung des Referenzmodells der Rotationsdynamik um Fehlerregler, PCH und neuronale Netze.

Da bei den verwendeten unbemannten Flugzeugen kein Seitenruder zur Verfügung steht und die Gierrate nicht direkt beeinflusst werden kann, ist der Gierraten-Fehlerregler und das neuronale Netz in der Gierratenregelung wirkungslos. Um CPU-Ressourcen zu sparen wird daher auf ein neuronales Netz in der Gierratenregelung verzichtet. Die Netzeingänge der Roll- und Nickratenregelung werden in Tabelle 5.1 und 5.2 zusammengefasst.

Bezeichnung	**Symbol**	Einheit
Rollrate	p_t, p_{t-1}	rad\ s
Referenzsignal-Rollrate	$p_{m,ref,t}$, $p_{m,ref,t-1}$	rad\ s
Rollwinkel	Φ_t, Φ_{t-1}	°
Nickwinkel	Θ_t	°
Nickrate	q_t	rad\ s
Gierrate	r_t	rad\ s
Staudruck	$\bar{q}$	N\m^2

Tabelle 5.1.: Eingangssignale $\underline{\vec{x}}_p$ der Rollratenregelung.

Da, wie bereits erwähnt, Kopplungseffekte in der Fehlerdynamik zu erwarten sind, können diese auch von den neuronalen Netzen berücksichtigt werden. Dazu wird, z.B. bei der Nickratenregelung, zusätzlich die Rollrate als Eingangssignal der neuronalen

Netze verwendet. Ferner ist die Rotationsdynamik stark vom Staudruck $\bar{p}$ abhängig, so dass dieses Signal auch einer Verringerung des Fehlers dient.

Bezeichnung	**Symbol**	Einheit
Nickrate	q_t, q_{t-1}	rad\ s
Referenzsignal-Nickrate	$q_{m,ref,t}$, $q_{m,ref,t-1}$	rad\ s
Nickwinkel	Θ_t, Θ_{t-1}	°
Rollwinkel	Φ_t	°
Rollrate	p_t	rad\ s
Gierrate	r_t	rad\ s
Staudruck	$\bar{q}$	N\m^2

Tabelle 5.2.: Eingangssignale $\underline{\vec{x}}_q$ der Nickratenregelung.

Die Dimensionen eines neuronalen Netzwerks werden nach [Rob13] zu 10 Eingangsneuronen und 20 Neuronen in der verdeckten Schicht gewählt. Mit Berücksichtigung der Bias-Verbindungen ergeben sich 241 Verbindungsgewichte des neuronalen Netzes. Damit kommen 482 Verbindungen für die Roll- und Gierratenregelung zusammen.

6 Adaptive Inversion in Versuchen

6.1 Simulation

In der Simulation können die stabilisierenden Effekte der Sliding-Mode-trainierten Netze beobachtet werden. Dazu wird das inzwischen mehrmals durchgeführte Experiment des Querruderfehlers ohne Parameteränderungen wiederholt (vgl. auch Anhang A). Ausführliche Untersuchungen im Rahmen von weiteren Modellfehlern, wie z.B. abweichenden Derivaten, werden im Zusammenhang mit dem unbemannten Luftfahrzeug T200 in [Krü12] ausführlich dokumentiert .

Betrachtet man den Bahn-, Nick- und Rollwinkelverlauf in Grafik 6.1, 6.2 und 6.3, so ist bis zum Einschaltzeitpunkt des Fehlers kein Unterschied zum Simulationsergebnis ohne neuronale Netze erkennbar.

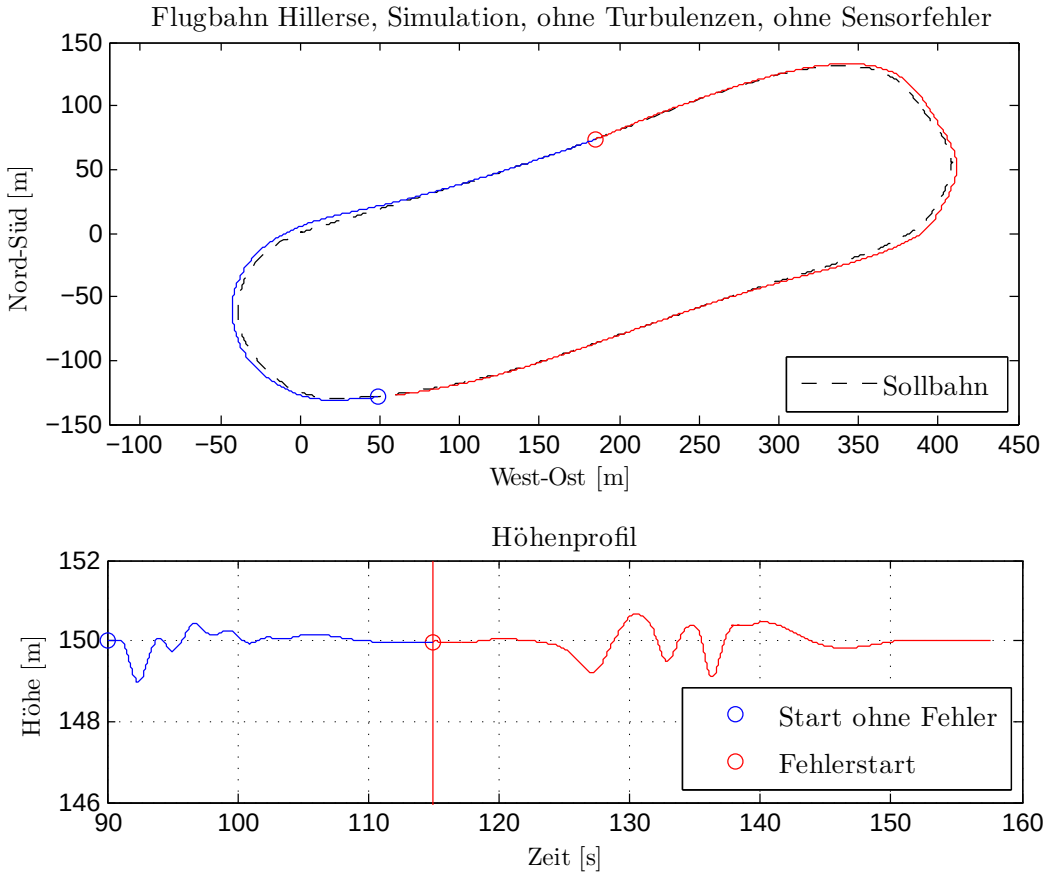


Abbildung 6.1.: Simulation mit neuronalen Netzen; Flugbahn und Flughöhe; Querruderfehler ab [t=115 s].

Ab dem Querruderfehler wird der positive Effekt der neuronalen Netze sichtbar, wobei das Flugzeug wesentlich unempfindlicher auf diesen Fehler reagiert. Bei Betrachtung des Bahn- und Höhenverlaufs ist ein Fehler zuerst nicht zu erkennen. Erst bei Betrachtung des Rollwinkelverlaufs im Fehlerzeitpunkt wird eine entstandene Abweichung sichtbar. Im weiteren Verlauf, also im Bereich [t=120 s ... 140 s] ist im Rollwinkel kein Unterschied zum Flugverhalten ohne Fehler zu erkennen.

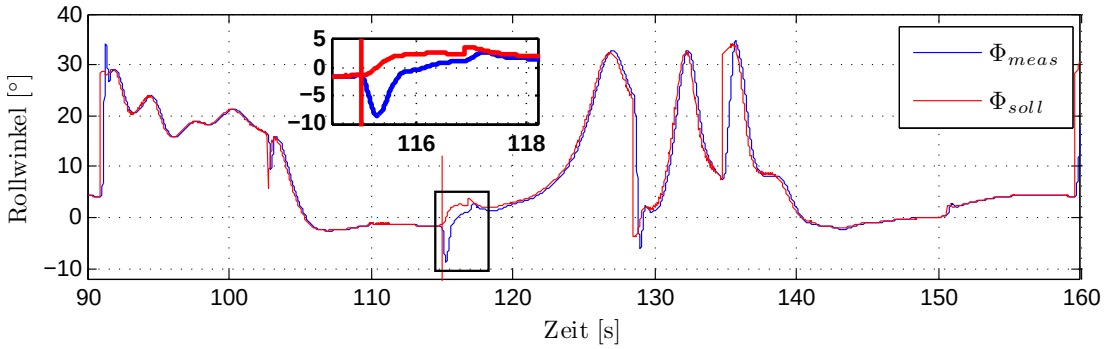


Abbildung 6.2.: Simulation mit neuronalen Netzen; gemessene und kommandierte Nickrate; Querruderfehler ab [t=115 s].

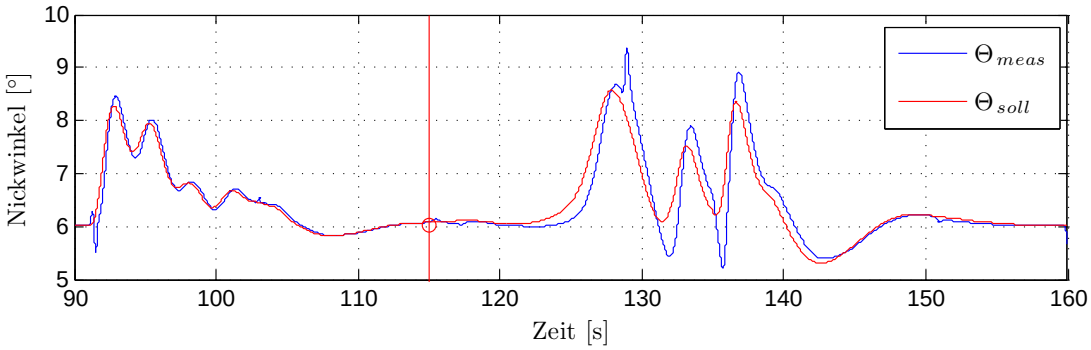


Abbildung 6.3.: Simulation mit neuronalen Netzen; gemessene und kommandierte Nicklage; Querruderfehler ab [t=115 s].

Am Netzwerkausgang $\nu_{ad,p}$ der Rollrate in Abbildung 6.4 ist das Verhalten der neuronalen Netze zu beobachten. Während am Anfang des Querruderfehlers der robustifizierende Term $\nu_{r,p}$ dominiert, kommt es innerhalb von $[\Delta t = 10s]$ zu einer Adaption des Querruderfehlers. Allerdings wird die Skalierung der Einheit $[°/s^2]$ im Fall des Inversionsfehlers ungültig. Die Ersatzregelgrößen $\nu_{i,p}$ repräsentieren eine kommandierte Drehbeschleunigung; im Fall eines Inversionsfehlers entspricht das erwartete Moment durch die halbierte Querruderwirksamkeit allerdings nicht mehr der

kommandierten Drehbeschleunigung. Das bedeutet, die neuronalen Netze geben eine „einheitenfreie“ Drehbeschleunigung vor, um den Inversionsfehler zu minimieren.

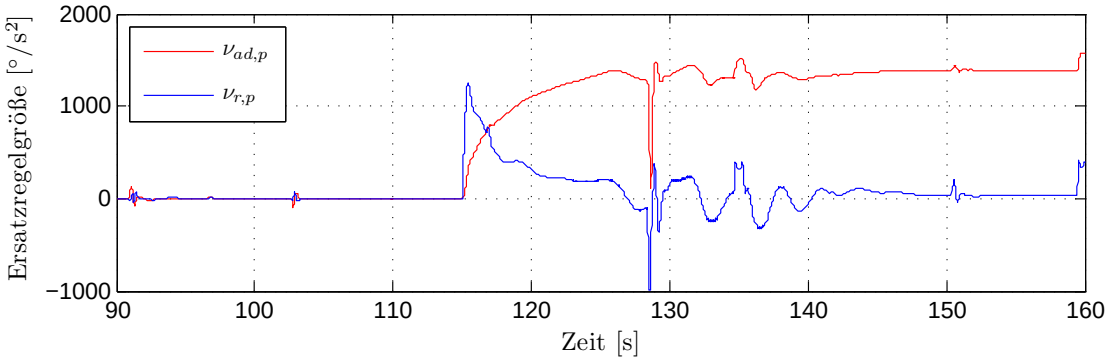


Abbildung 6.4.: Simulation mit neuronalen Netzen; Ausgang neuronale Netze $\nu_{ad,p}$ und robustifizierender Term $\nu_{r,p}$; Querruderfehler ab [t=115 s], ab Fehler Skalierung für Einheit [$°/s^2$] ungültig.

Der Regelfehler e_p in Abbildung 6.5 entspricht im Verlauf exakt dem robustifizierenden Term. Dessen Einheit bleibt allerdings, da er aus Soll- und Ist-Drehraten gebildet wird, gültig.

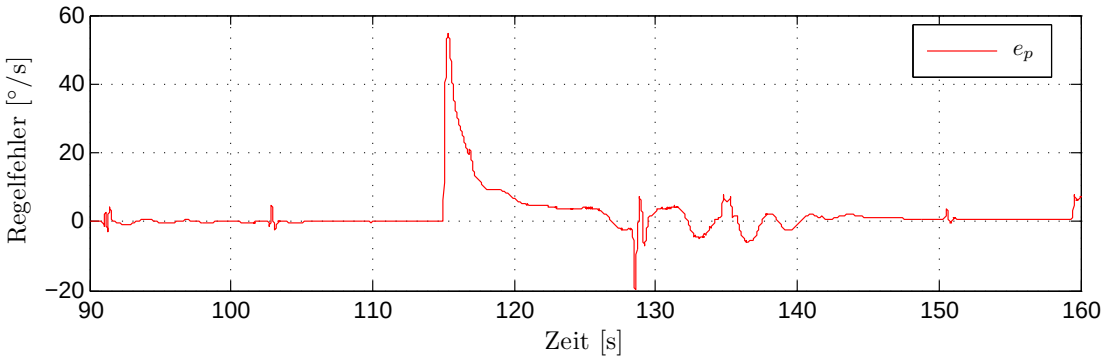


Abbildung 6.5.: Simulation mit neuronalen Netzen; Regelfehler e_p; Querruderfehler ab [t=115 s].

Die gemessene Rollrate p_{meas} wird in Abbildung 6.6 dem kommandierten Wert p_{soll} gegenübergestellt. Der Effekt des Querruderfehlers ist auch hier gut erkennbar. Die Abweichungen im Zeitbereich [t=120 s .. 140 s] sind allerdings nicht vollständig auf einen Inversionsfehler zurückzuführen, da der kommandierte Sollwert mit dem Referenzsignal abgebremst wird, wie in Abbildung 6.7 zu sehen. Da der Regelfehler als Differenz zwischen Referenzsignal und Messwert definiert wird, ist hier der Vergleich zwischen Soll- und Istwert der Drehrate kein optimaler Indikator von Regelgüte.

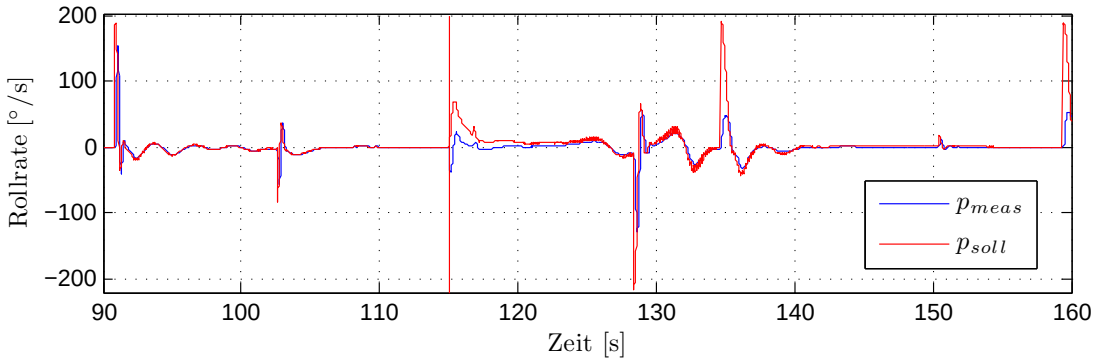


Abbildung 6.6.: Simulation mit neuronalen Netzen; gemessene Rollrate p_{meas} und vorgegebene Drehrate p_{soll}; Querruderfehler ab [t=115 s].

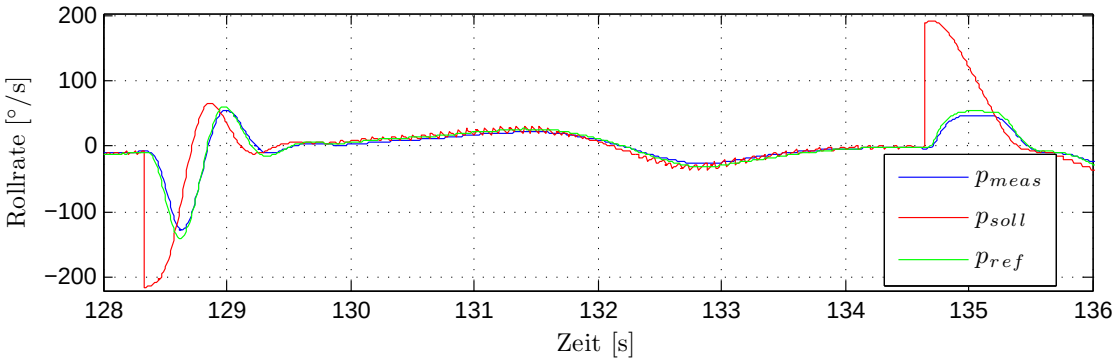


Abbildung 6.7.: Simulation mit neuronalen Netzen; gemessene Rollrate p_{meas}, Referenzsignal Rollrate p_{ref} und vorgegebene Drehraten p_{soll}.

Zusammenfassend kann man sagen, dass die neuronalen Netze in der Lage sind, den Inversionsfehler abzufangen und die Stabilität des Flugzeuges zu verbessern. Im Vergleich dazu führen die zuvor vorgestellten Strukturen ohne adaptive Elemente und ohne Parameteranpassungen nicht zu einer zufriedenstellenden Regelgüte.

Diskussion:
Die Tatsache, dass die Skalierung der Ersatzregelgröße nicht mehr stimmt, führt zu einem Ansatz für die zukünftige Weiterentwicklung des Inversionsreglers. Da zum Ausgleich des Regelfehlers ein skalenbehafteter Drehgeschwindigkeitsfehler messbar wird, allerdings eine unerwartet höhere Drehbeschleunigung kommandiert werden muss, kann hieraus das benötigte Moment und letztendlich der Beiwert des Querruders abgeleitet werden. Es stellt sich die Frage, ob dieser Beiwert dabei nicht vom neuronalen Netz erlernt werden kann. Damit besteht die Möglichkeit, die Ausgänge von neuronalen Netzen als Beiwerte zu nutzen, sofern Freiheitsgrade berücksichtigt werden. Einen ähnlichen Ansatz verfolgt bereits [NS07].

6.2 Hardware in the Loop

Bevor die adaptive dynamische Inversion im Flugversuch untersucht wird, soll das CPU- und Softwareverhalten im HIL- (Hardware in the Loop) Verfahren analysiert werden. Mit dieser Untersuchung kann das Echtzeitverhalten, also die Einhaltung des Rechenzeit-Rahmens von 10 ms bei 100 Hz Regeltakt, bestimmt werden. Bei einer HIL-Simulation werden die Sensordaten aus einer Simulation an den Regler-Algorithmus auf der Zielhardware (Autopilot) gesendet. Die Aktuatorsignale des Autopiloten werden wiederum in der Simulation verarbeitet, womit der Kreis (Loop) geschlossen wird. Ein Vorteil der hohen Prozessorkapazitäten des im Autopiloten verwendeten Prozessors OMAP-3530 liegt darin, dass die gesamte Flugzeugsimulation, inklusive der Regelungs- und Navigationsstrukturen auf dem Autopiloten ausgeführt werden kann. Dies ermöglicht eine HIL-Simulation ohne einen zweiten Simulationscomputer.

Die Implementierung der gesamten Simulations-Navigations-Reglerstruktur erfolgt mit dem in Abbildung 6.8 dargestellten Simulink-Schaltplan. Dieser Schaltplan wird direkt in Simulink simuliert oder mit einem „Maus-Klick“ in den Autopiloten programmiert. Mit dem Schalter „HIL-Switch“ wird dieser Schaltplan auf dem Autopiloten entweder im Hardware in the Loop Verfahren untersucht oder direkt für den Flugversuch verwendet. Innerhalb des Subsystems „Regler“ sind alle drei Reglervarianten, „Basisregler“, „dynamische Inversion“ und „adaptive dynamische Inversion“ enthalten. Je nach Wunsch des Anwenders wird zwischen den Reglern umgeschaltet. Folgend werden die wichtigsten Eckdaten des Schaltplans dargestellt:

- Anzahl Subsysteme im Schaltplan: 137
- Codegenerierung: 31.834 Zeilen, bzw. 1.58 MB C-Code
- Ausführbare Binärdatei: 317 kB
- Speicherverbrauch OMAP-3530: 58 kB

Zur weiteren Untersuchung der HIL-Eigenschaften wird folgender Versuchsaufbau generiert: Während der Spline-Regler inkl. Navigation den Lagewinkel zur Bahnfolge generiert, werden die übrigen Reglerstrukturen in drei Stufen ausgeführt. Zuerst wird der Basisregler zur Bahnregelung ausgeführt. Nach ca. 50 Sekunden übernimmt die dynamische Inversion ohne neuronale Netze die Kontrolle. Bei ca. [t=90 s] werden die Sliding-Mode trainierten neuronalen Netze hinzugeschaltet. Die Messung der CPU-Auslastung erfolgt mit dem Linux-Programm „top“. Die CPU-Auslastung in

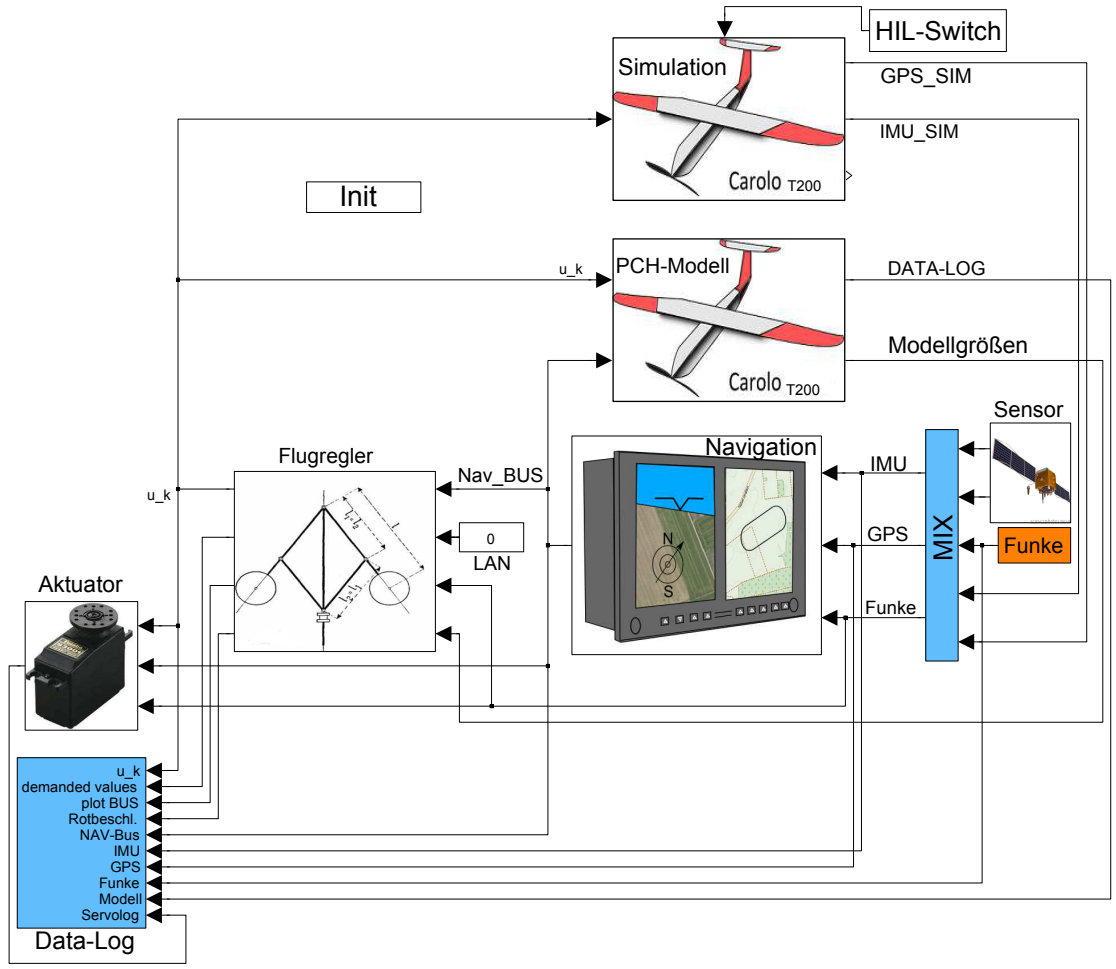


Abbildung 6.8.: Simulink Schaltplan.

diesem Versuch, in Abbildung 6.9 dargestellt, variiert je nach Reglertyp. Die ersten 8 Sekunden dieses Diagramms haben keine Aussage. Bei etwa [t=50 s] ist mit der Umschaltung vom Basis- zum Inversionsregler ein leichter Anstieg der Prozessorauslastung um ca. 1% zu erkennen. Mit Zuschaltung der neuronalen Netze bei [t=90 s] steigt der CPU-Verbrauch auf 46%. Der SPI-Treiber (SPIO) und nicht echtzeitfähige Prozesse verwenden ca. 2% der CPU .

Die Aussagekraft von „top“ ist allerdings begrenzt. „top“ wird als nicht echtzeitfähiger Prozess ausgeführt und mittelt jede Sekunde die Zeit der Prozessor-Belegung durch die im Betriebssystem ausgeführten Prozesse. Da weitere, nicht echtzeitfähige Prozesse auch das Programm „top“ blockieren können, kommt es hierbei zu Messfehlern und dem Aussetzen von Messungen. Die Folge sind Schwankungen in der CPU-Auslastungs-Messung sowie eine fehlerhafte Zeitskala gegenüber der Simulation. Die CPU-Auslastung ist zwar ein gutes Indiz für die Ausführungsdauer, bedingt durch

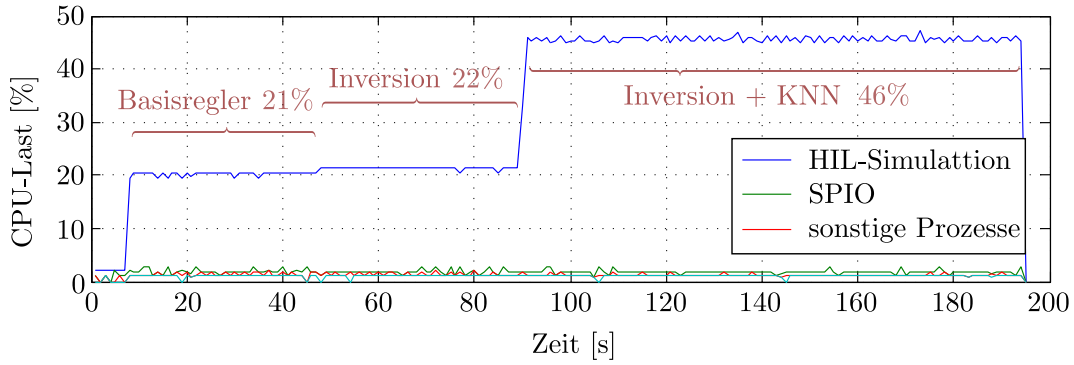


Abbildung 6.9.: Hardware in the Loop: CPU-Auslastung, verschiedene Regler auf dem OMAP-3530.

die Mittelung kann hierdurch allerdings noch keine Aussauge zur Echtzeitfähigkeit getroffen werden. Um die Problematik genauer zu veranschaulichen, wird hierzu in Abbildung 6.10 die Thread-Dauer der HIL-Simulation dargestellt. Die Thread-Dauer ist ein exaktes Maß der Ausführungszeit eines Simulation-Reglertaktes. Auf den ersten Blick scheint diese Ausführungszeit zu „rauschen“.

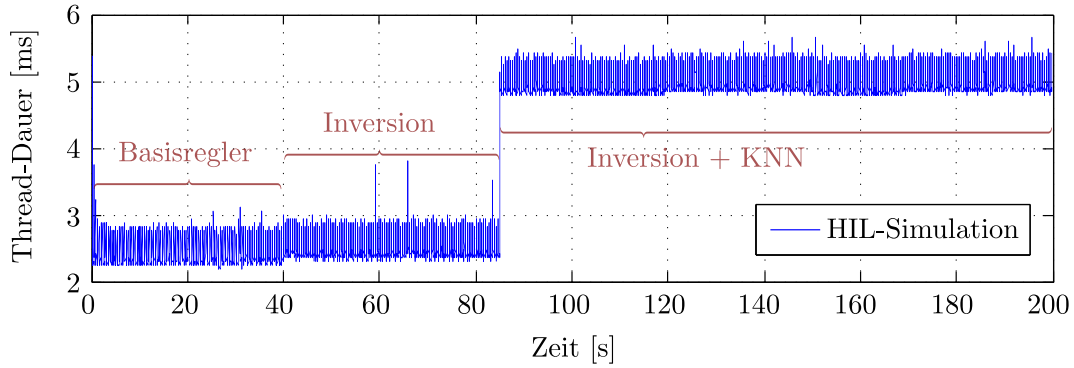


Abbildung 6.10.: Hardware in the Loop: Thread-Dauer des Simulation-Reglertaktes.

Bei genauerer Betrachtung dieses „Rauschens“ in Abbildung 6.11 wird der Zusammenhang zur Navigation deutlich: Wie bereits im Unterkapitel 3.3 erwähnt, ist der Kalman Filter aus einem Update und einer Propagation aufgebaut. Alle 25 ms wird mit einer GPS-Messung ein Update ausgelöst, was zu einem erhöhten Rechenzeitbedarf führt. Zudem ist der erhöhte Rechenzeitbedarf während der Initialisierungsphase des Filters sichtbar.

Abschließend werden die verfügbaren Integrationsverfahren im Zusammenhang der dynamischen Inversion in Grafik 6.12 dargestellt. Alle Simulations- und Flugversuche, die im Rahmen dieser Arbeit dargestellt werden, basieren auf dem Runge-Kutta

(ode4) Verfahren. Allerdings sind mit den zur Verfügung stehenden Ressourcen auch andere Integrationsverfahren denkbar. Inwieweit eine Variation der Integrationsverfahren die Regelgüte verbessert oder verschlechtert, bleibt zu untersuchen.

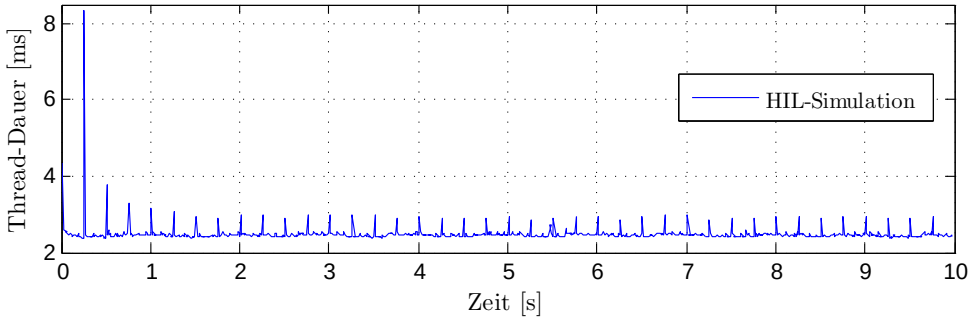


Abbildung 6.11.: Hardware in the Loop: Echtzeit-Test der Simulation auf dem OMAP-3530, Kalman Filter Updates berechnen länger.

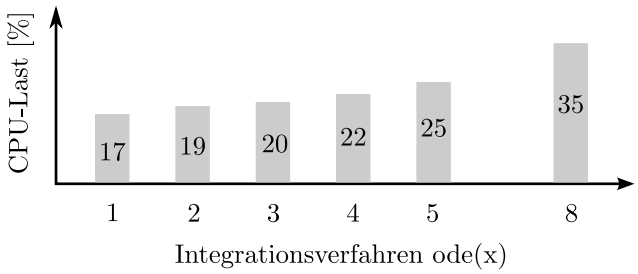


Abbildung 6.12.: Hardware in the Loop: CPU-Last mit verschiedenen Integrationsverfahren, dynamische Inversion ohne neuronale Netze.

6.3 Instabilität im Flugversuch

Im Flugversuch zeigt sich ein anderes Verhalten der neuronalen Netze gegenüber der Simulation aus Unterkapitel 6.1, wobei ein stetig wachsendes Rauschen an deren Ausgang $\nu_{ad,i}$ sichtbar wird. Ferner erreichen die neuronalen Netze eine Grenze ihrer internen Resonanzkatastrophe, wie man an Abbildung 6.13 bei $[t = 8.18\ \text{s}]$ erkennen kann.

Um diesen Effekt nachzubilden, ist es daher sinnvoll, das Sensorrauschen der Gyroskope zu berücksichtigen. Die Dauer bis zum Erreichen der Resonanzkatastrophe hängt allerdings von der Rauschintensität ab, wie in den Abbildungen 6.14 sichtbar

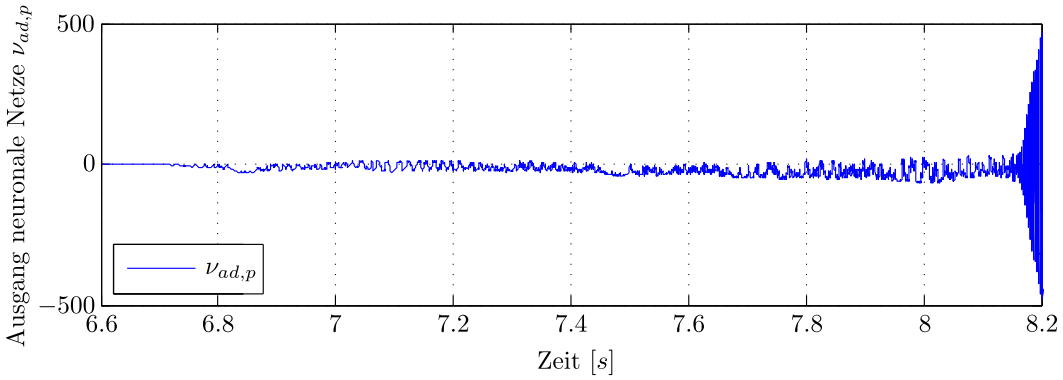
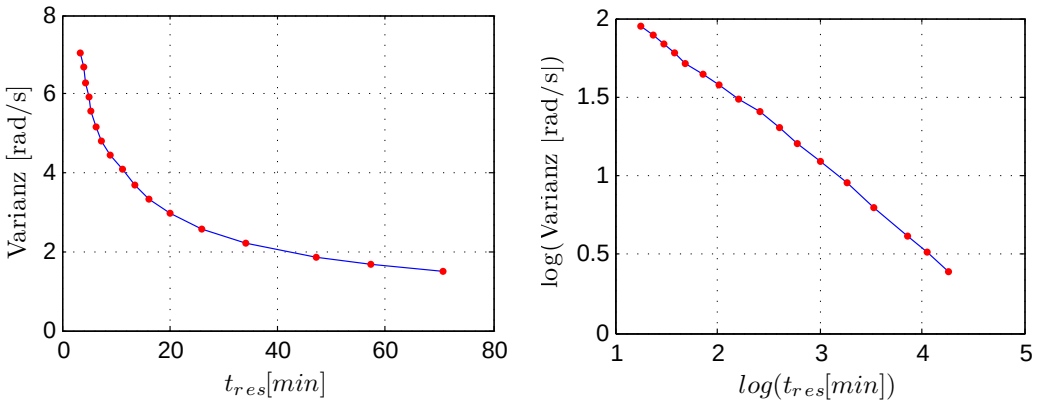


Abbildung 6.13.: Flugversuch mit neuronalen Netzen; Rauschen im Drehratensensor führt zu instabilem Ausgang des neuronalen Netzes.

wird. Diese Grafiken wurden mit dem Sliding-Mode-Trainingsverfahren und weißem Rauschen mit einer Bandbreite von 100 Hz erstellt. Die logarithmische Darstellung dieses Diagramms zeigt eine Gerade, was die These stützt, dass die Netze mit Sensorrauschen immer instabil werden. Beim klassischen Gradientenabstiegs-Lernverfahren ist dasselbe Verhalten zu beobachten. Der Resonanzeffekt ist hier auch von der Lernrate μ abhängig und wird von dieser verzögert oder beschleunigt.



(a) Dauer bis zur Resonanzkatastrophe. (b) Log. Dauer bis zur Resonanzkatastrophe.

Abbildung 6.14.: Simulationsversuche; weißes Rauschen in den Drehraten führt nach bestimmter Zeit zur Destabilisierung der neuronalen Netze in Abhängigkeit der Rauschintensität.

Das dargestellte Phänomen ist dahingehend von gefährlicher Natur, da möglicherweise eine vorgenommene Simulation bzw. ein Flugversuch mit einem moderaten Sensorrauschen modelliert bzw. durchgeführt werden kann, z.B. $\mathrm{Var}(\omega_{noise}) = 0.1$ deg/s. Die Resonanzkatastrophe wird hierdurch aber erst nach Stunden eintreten.

6.4 E-Modifikation und Stabilität

Mit dem letzten Flugversuch in der Simulation konnte bereits dargestellt werden, dass die neuronalen Netze einen stabilisierenden Effekt im Vorhandensein von Inversionsfehlern auf die dynamische Inversion haben. Dennoch kann die bisherige Konfiguration noch nicht im realen Flugversuch angewendet werden, da die neuronalen Netze offensichtlich durch Sensorrauschen sehr schnell einen kritischen Punkt ihrer inneren Resonanzkatastrophe erreichen. So ist es an dieser Stelle unumgänglich, die Thematik der Stabilität des Systems genauer zu betrachten. Bereits in [JC00] wurde das Thema der Stabilität mit neuronalen Netzen, dynamischer Inversion und PCH bearbeitet. Die allgemeine Stabilitätsbetrachtung eines geschlossenen Regelkreises mit neuronalen Netzen ist auf [LYL96] zurückzuführen. Dabei wird erwähnt, dass die Gewichte des Gradientenabstiegs-Lernverfahrens, im Vorhandensein von nicht rekonstruierbaren Funktionen oder unbekannten Störungen, unbegrenzt wachsen könnten. Um diesem Effekt entgegenzuwirken wurde mit [LYL96] die sogenannte E-Modifikation eingeführt.

Dabei wird die klassische Rückpropagation aus Gleichung (5.27) um einen weiteren Term erweitert:

$$\Delta \mathbf{w} = \mu \cdot \frac{\partial E(\mathbf{w}, \vec{x})}{\partial \mathbf{w}} - \lambda \cdot \|\zeta\|_2 \cdot \mathbf{w}. \tag{6.1}$$

Dieser Term führt, in Abhängigkeit des Parameters λ, zu einem langsamen Abnehmen der Netzwerkgewichte, hin zum Gleichgewichtspunkt des Netzwerkes. Dieser Term wird in [LEWIS39TK] auch treffend als *Term des Vergessens* bezeichnet.

Nach [Krü12] kann diese Erweiterung auch auf das Sliding-Mode-Lernverfahren (vgl. Gleichung (6.2)) angewendet werden:

$$\Delta \mathbf{w} = \left(\frac{\partial \vec{y}(\mathbf{w}, \vec{x})}{\partial \mathbf{w}} \right)^T \mu \cdot diag(sign(S_t)) \cdot |\vec{\epsilon}| - \lambda \cdot \mu \cdot \|\zeta\|_2 \cdot w_{t-1,ij}. \tag{6.2}$$

Mit der E-Modifikation ist [Hol04] der Stabilitätsbeweis für die dynamische Inversion im Rahmen des Gradientenabstiegs-Lernverfahrens gelungen. Im Rahmen des Sliding-Mode-Lernverfahrens mit E-Modifikation wird in [Krü12] die Stabilitätsuntersuchung vorgestellt. Die Darstellung des Resonanzkatastrophenproblems im Zusammenhang mit der E-Modifikation wird dennoch als wichtig erachtet, da hier zum ersten Mal ein praktisches Beispiel der Notwendigkeit dieser Modifikation im geschlossenen Kreis der dynamischen Inversion dargestellt wird.

6.5 Adaptive Inversion im Flugversuch

Mit der E-Modifikation ist es im Rahmen dieser Arbeit erstmals im Flugversuch gelungen, die dynamischen Inversion mit neuronalen Netzen im Flugversuch stabil zu betreiben, wie in Abbildung 6.15 dargestellt. Die folgenden Darstellungen basieren auf Sliding-Mode trainierten Netzen. Bereits im Unterkapitel 4.9.2 wurde dargestellt, dass es trotz einer fehlerhaften dynamischen Inversion möglich ist, das Flugzeug weiterzufliegen. Allerdings entstehen unerwünschte Bahn- und Lageabweichungen aus den Inversionsfehlern. Um einen Zusammenhang zu den vorherigen Ergebnissen herzustellen, ist der folgend dargestellte Flugversuch eine Fortführung des Flugversuchs im Unterkapitel 4.9.2. Eine Veränderung der Reglerparameter findet nicht statt, es werden nur neuronale Netze hinzu geschaltet und RCH auf PCH umgestellt.

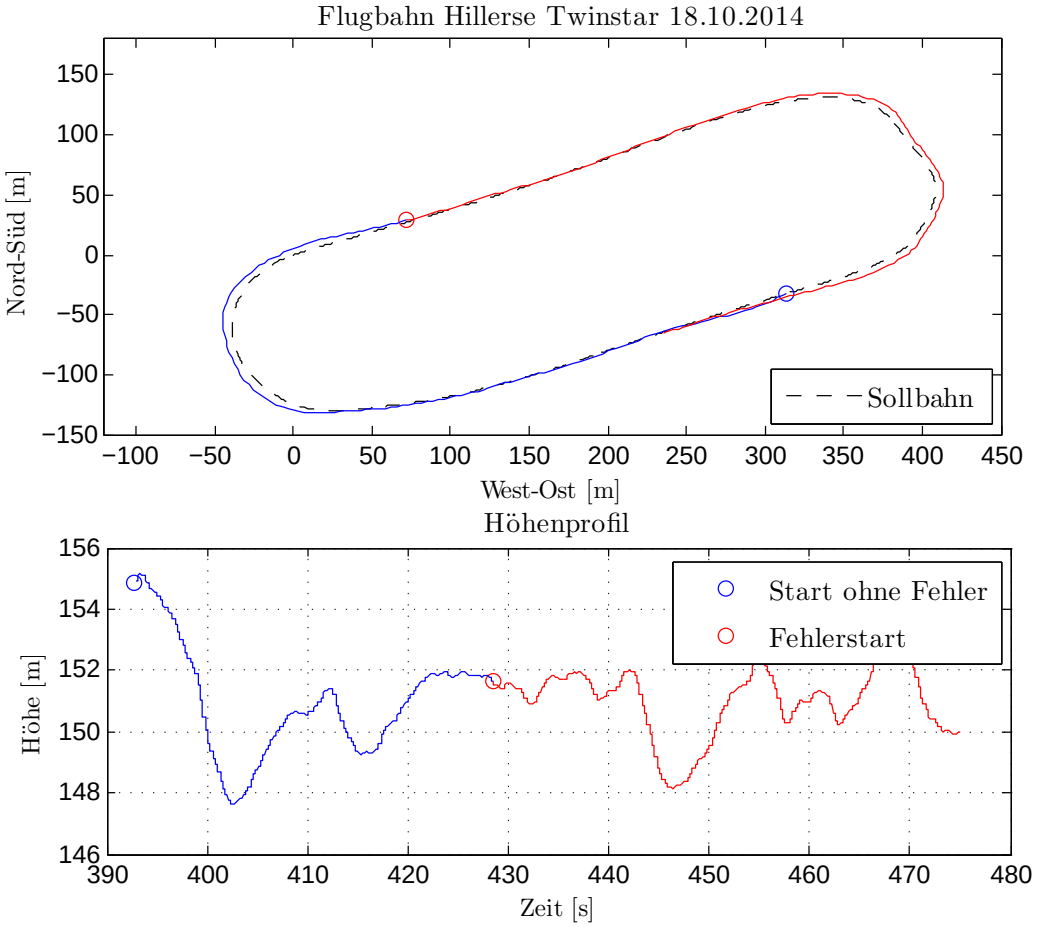


Abbildung 6.15.: Flugversuch mit KNN; GPS-Bahnverlauf und Höhe; blauer Kreis: ohne Fehler; roter Kreis: mit blockiertem linken Querruder $\xi_{li} = 7°$.

Der Flugversuch ist wie folgt aufgebaut: Noch während des Fluges mit der dynamischen Inversion (vlg. dazu Grafik 4.28 bis 4.31) und abgeschaltetem Querruderfehler werden die neuronalen Netze eingeschaltet. Nach ca. 35 Sekunden wird zusätzlich der Querruderfehler aktiviert. Bereits mit dem Einschalten der neuronalen Netze ist eine deutliche Verbesserung der Flugeigenschaften gegenüber dem Regler ohne adaptive Elemente festzustellen, Bahn- oder Lageabweichungen sind mit neuronalen Netzen bis zum Einschaltzeitpunkt des Querruderfehlers nicht mehr zu erkennen. Der Unterschied in der Lageregelung mit und ohne adaptive Elemente wird im Roll- und Nickwinkelverlauf in Abbildung 6.16 und 6.17 deutlich.

Im Gegensatz zur dynamischen Inversion ohne neuronale Netze wird der Regelfehler e_p in der Rollrate abgebaut, wie anhand von Abbildung 6.18 und 6.19 sichtbar wird. Der Ausgang des neuronalen Netzes $\nu_{ad,p}$ und der robustifizierende Term $\nu_{r,p}$ in Grafik 6.20 verhalten sich ähnlich wie in der Simulation (vgl. Grafik 6.4) vorhergesagt. Die Grafiken 6.23 bis 6.27 stellen die Aktivität der Nickratenregelung dar, wobei im Gegensatz zur Simulation (Anhang A) ein leicht verändertes Verhalten festzustellen ist.

Die Wirkung des neuronalen Netzes wird zudem im Aktuatorsignal in Abbildung 6.21 sichtbar: Während ohne Querruderfehler relativ kleine Aktuatorsignale das Bild dominieren, wird nach dem Einschalten des Fehlers eine deutlich höhere Aktuatoraktivität erkennbar.

Bedenkt man die modellierte Querruderbegrenzung von $\xi_{max} = \pm 20°$, so wird deutlich, dass es im Zeitbereich von t=[450 s ... 470 s] zur Aktuatorbegrenzung und folglich einer Reaktion des PCH kommen muss. Auch schnelle Aktuatorsignale werden durch die Aktuatordynamik begrenzt und führen zu einem Hedgesignal $\nu_{h,\dot{p}}$ im Rollsignal. Beide Effekte werden in den Aufzeichnungen in Abbildung 6.22 sichtbar.

Zusammenfassend kann man sagen, dass die neuronalen Netzen einen hohen Beitrag zur Verbesserung der Flugeigenschaften liefern. Trotz Modellfehlern ist eine Bahnführung sehr gut möglich. Ein sprungartiges Zuschalten einer Degradation, in Form eines Querruderfehlers, wird erst bei Betrachtung des Lageverlaufs sichtbar. Die Wiederholung des Experiments führt immer wieder zum gleichen Ergebnis.

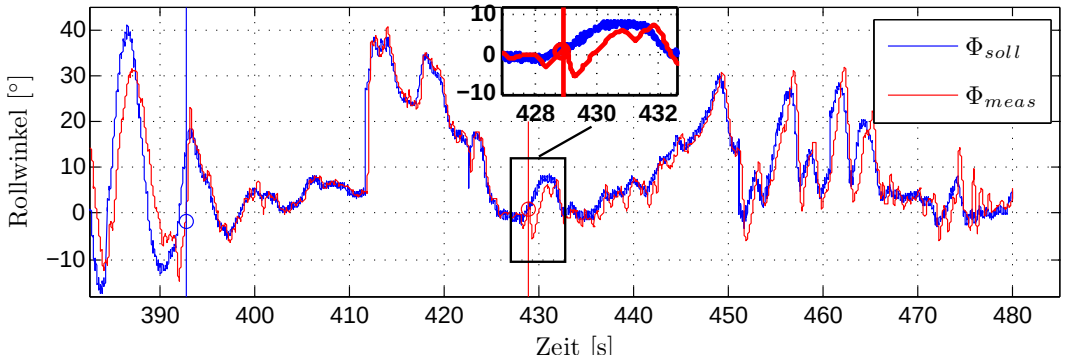


Abbildung 6.16.: Flugversuch; Rollwinkel Ist- und Sollverlauf; blaue vert. Linie: neuronale Netze einschalten; rote vert. Linie: blockiertes linkes Querruder $\xi_{li} = 7°$.

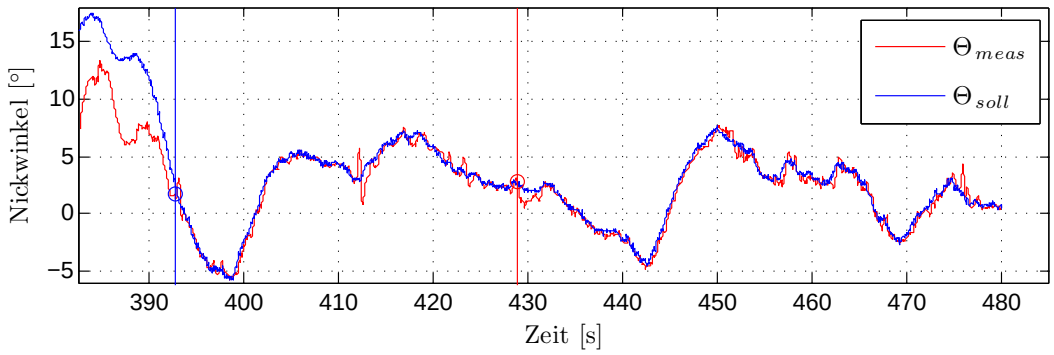


Abbildung 6.17.: Flugversuch; Nickwinkel Ist- und Sollverlauf; blaue vert. Linie: neuronale Netze einschalten; rote vert. Linie: blockiertes linkes Querruder $\xi_{li} = 7°$.

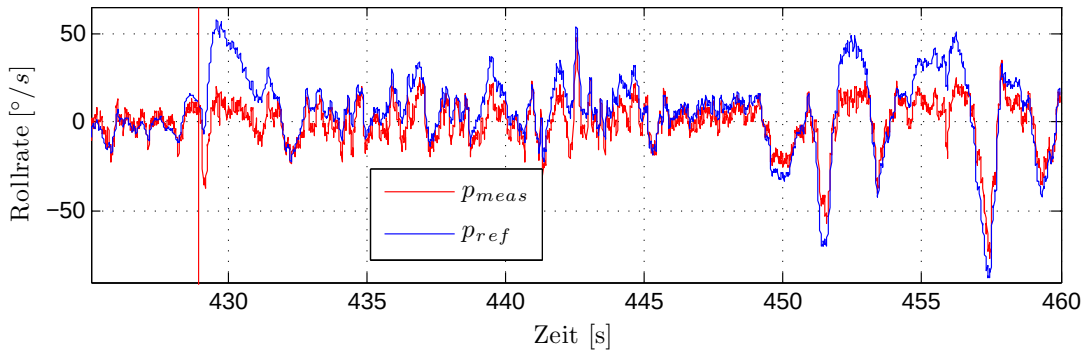


Abbildung 6.18.: Flugversuch; Rollrate Ist- und Sollverlauf; rote vert. Linie: blockiertes l. Querruder $\xi_{li} = 7°$.

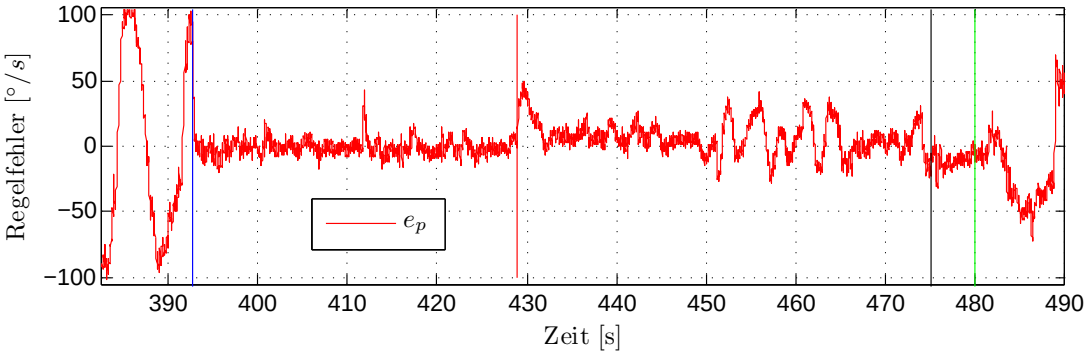


Abbildung 6.19.: Flugversuch; Regelfehler Rollregler; blaue vert. Linie: neuronale Netze einschalten; rote vert. Linie: blockiertes l. Querruder $\xi_{li} = 7°$; schwarze vert. Linie: Blockade deaktiviert; grüne vert. Linie: neuronale Netze deaktiviert.

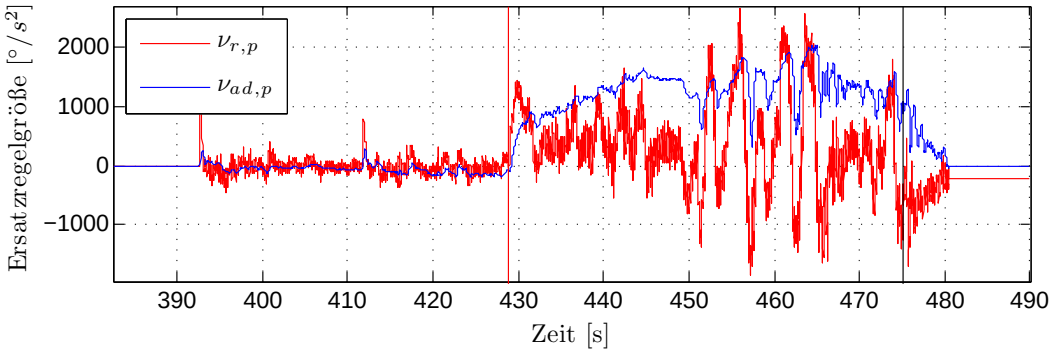


Abbildung 6.20.: Flugversuch; robustifizierender Term und Ausgang neuronale Netze; rote vert. Linie: blockiertes l. Querruder $\xi_{li} = 7°$; schwarze vert. Linie: Blockade deaktiviert.

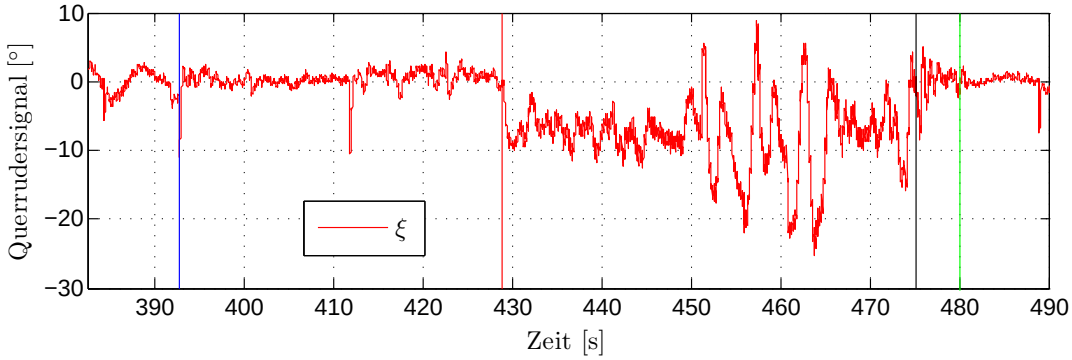


Abbildung 6.21.: Flugversuch; Querrudersignal; blaue vert. Linie: neuronale Netze einschalten; rote vert. Linie: blockiertes l. Querruder $\xi_{li} = 7°$; schwarze vert. Linie: Blockade deaktiviert; grüne vert. Linie: neuronale Netze deaktiviert.

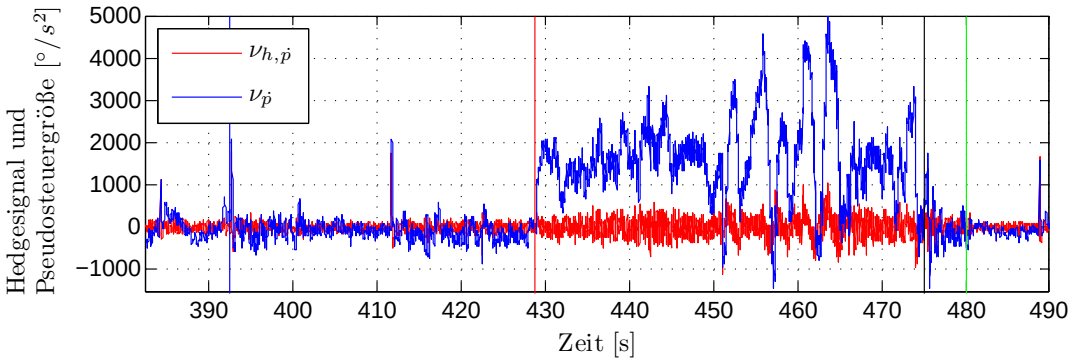


Abbildung 6.22.: Flugversuch; Hedgesignal und Pseudosteuergröße Rollregler; blaue vert. Linie: neuronale Netze einschalten; rote vert. Linie: blockiertes l. Querruder $\xi_{li} = 7°$; schwarze vert. Linie: Blockade deaktiviert; grüne vert. Linie: KNN deaktiviert.

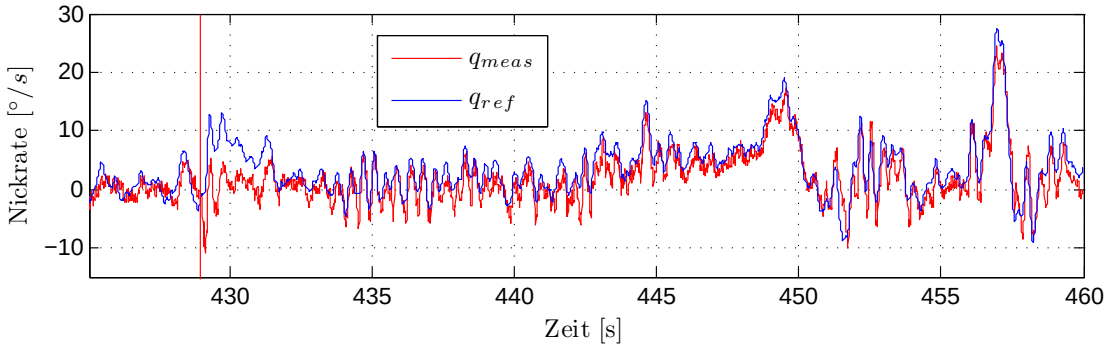


Abbildung 6.23.: Flugversuch; Nickrate Ist- und Sollverlauf; rote vert. Linie: blockiertes l. Querruder $\xi_{li} = 7°$.

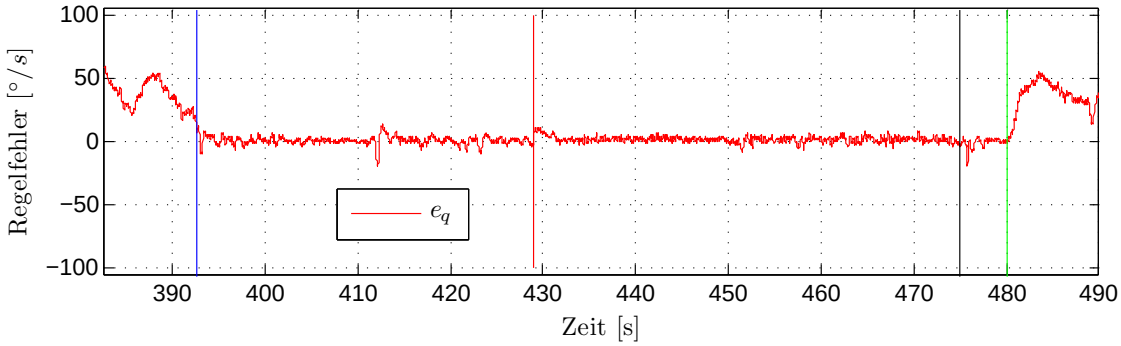


Abbildung 6.24.: Flugversuch; Regelfehler Nickregler; blaue vert. Linie: neuronale Netze einschalten; rote vert. Linie: blockiertes l. Querruder $\xi_{li} = 7°$; schwarze vert. Linie: Blockade deaktiviert; grüne vert. Linie: neuronale Netze deaktiviert.

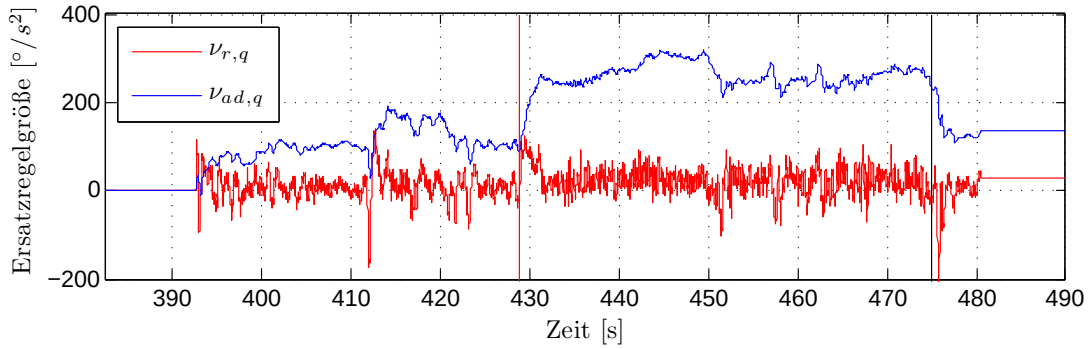


Abbildung 6.25.: Flugversuch; robustifizierender Term und Ausgang neuronale Netze Nickregler; rote vert. Linie: blockiertes l. Querruder $\xi_{li} = 7°$; schwarze vert. Linie: Blockade deaktiviert.

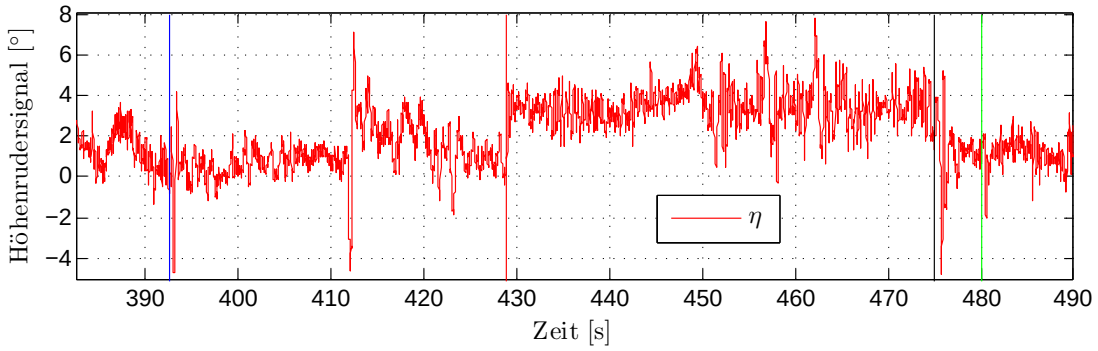


Abbildung 6.26.: Flugversuch; Höhenrudersignal; blaue vert. Linie: neuronale Netze einschalten; rote vert. Linie: blockiertes l. Querruder $\xi_{li} = 7°$; schwarze vert. Linie: Blockade deaktiviert; grüne vert. Linie: KNN deaktiviert.

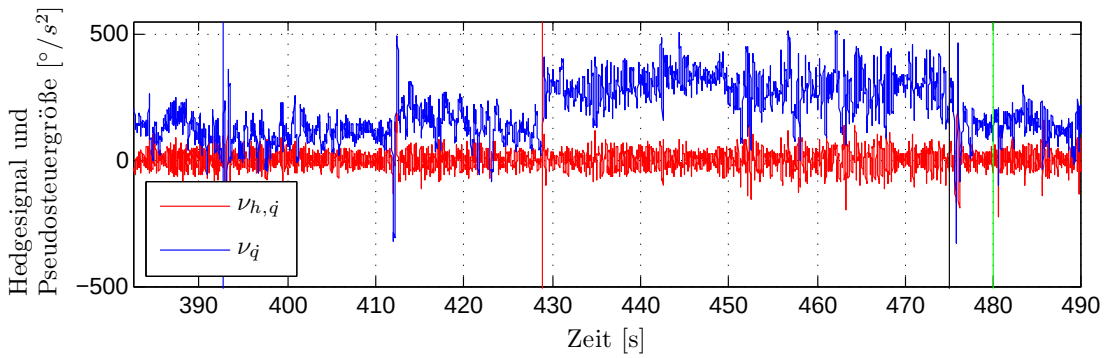


Abbildung 6.27.: Flugversuch; Hedgesignal und Pseudosteuergröße Nickregler; blaue vert. Linie: neuronale Netze einschalten; rote vert. Linie: blockiertes l. Querruder $\xi_{li} = 7°$; schwarze vert. Linie: Blockade deaktiviert; grüne vert. Linie: KNN deaktiviert.

7 Zusammenfassung und Ausblick

7.1 Zusammenfassung

In der vorliegenden Arbeit wird die Verknüpfung der nichtlinearen adaptiven Regelung mit der praktischen Umsetzung in unbemannten Flugversuchen vorgestellt. Das Ziel der Untersuchungen ist der Vergleich eines klassischen Linearreglers, eines nichtlinearen Reglers und eines adaptiven nichtlinearen Reglers mit neuronalen Netzen im Degradationsfall während des Flugversuchs. Gleichzeitig wird die Implementierung und das Verhalten der Regleralgorithmen auf der Autopilotenhardware betrachtet.

Beginnend mit dem Autopilotensystem wird eine Methode zur Umsetzung der Flugversuche vorgestellt. Dabei werden die Einflüsse der Hard- und Softwarekomponenten auf das Systemverhalten in den Regleralgorithmen berücksichtigt und die Bedeutung der Implementierungskette hervorgehoben. Dabei zeigt sich, dass die Konfiguration des Betriebssystems sowie die Anordnung der Sensor- und Aktuatorstruktur zu Verzögerungen im Gesamtsystem beitragen. Die Wahl der Implementierungskette erlaubt es, die gesamte Simulation inkl. Navigation, Bahnregelung und aller Reglerarchitekturen direkt aus einem Schaltplan in den Autopiloten zu programmieren.

Im Kern der Arbeit steht die ausführliche Darstellung des Simulationsmodells, der verwendeten Regleralgorithmen sowie der Aufbau der neuronalen Netze. Parallel zu den Reglerarchitekturen werden wiederholt die Ergebnisse von Simulationen und Flugversuchen betrachtet, um einen Vergleich des jeweiligen Systemverhaltens im Degradationsfall zu gewährleisten. Die Verwendung einer Bahnregelung unterstützt die Vergleichbarkeit der Flugversuche. Da sich die dynamische Inversion mit Pseudo-Control-Hedging ohne adaptive Elemente im Fall von Inversionsfehlern als unpraktikabel für Flugversuche herausstellt, wird mit dem Reference-Control-Hedging eine mögliche Variation vorgestellt, die einen Vergleich der dynamischen Inversion mit und ohne adaptive Elemente vereinfacht. Bei den Versuchen wird auch Wert auf das Echtzeitverhalten der Regleralgorithmen und der neuronalen Netze gelegt, um eine Abschätzung der Implementierbarkeit für weiterführende Forschungsprojekte zu ermöglichen.

Eine wichtige Rolle spielt in dieser Arbeit das Sliding-Mode-Lernverfahren, das wesentliche Vorteile gegenüber anderen Lernverfahren bietet. Neben besseren Adaptionseigenschaften wird mit diesem Verfahren die Betrachtung der neuronalen Netze als zu regelndes System ermöglicht. Die Ergebnisse dieser Arbeit zeigen, dass dieses Lernverfahren trotz besserer Eigenschaften im Vergleich zum einfachen Gradientenabstiegs-Lernverfahren gleiche Ressourcenanforderungen stellt. Das Sliding-Mode-Lernverfahren wird im Rahmen dieser Arbeit erstmals im Flugversuch erprobt, womit die Vorhersagen aus Simulationsversuchen bestätigt werden können.

Modellunsicherheiten dienen in dieser Arbeit als Mittel der Darstellung und der Verifikation von Eigenschaften der neuronalen Netze. Auch bei fehlerhaftem Inversionsmodell und zusätzlichem sprungförmigen Querruderfehler ist mit den adaptiven Strukturen, im Vergleich zu nicht adaptiven Strukturen, eine deutlich verbesserte Bahn- und Lagefolge zu erkennen. Die Betrachtung der Reaktion auf eine sprungförmige Degradation hebt die Bedeutung von adaptiven Elementen in Regelungsstrukturen besonders hervor. In Anbetracht der gezeigten Fähigkeiten der neuronalen Netze, im Rahmen regelungstechnischer Anwendungen und im Auftreten von Modellunsicherheiten, wird die Relevanz der weiteren Erforschung von künstlicher Intelligenz sichtbar.

Abschließend kann festgestellt werden, dass die wissenschaftliche Erforschung und Erprobung der künstlichen Intelligenz in Flugversuchen ohne Programmierkenntnisse möglich ist. Es zeigt sich, dass ein FPGA für die adaptive dynamische Inversion nicht zwingend notwendig ist, was die Implementierung vereinfacht. Die kostengünstige Nutzung von „Spielzeug“-Modellflugzeugen und Smartphonetechnik erlaubt einen weiten Einsatz eines einheitlichen Fluggerätesystems. Zuletzt muss betont werden, dass die adaptive nichtlineare Regelung mit neuronalen Netzen ein beachtliches Potenzial zur Verbesserung konservativer Regelungsstrukturen aufweist.

7.2 Ausblick

Die Untersuchungen im Rahmen dieser Arbeit zeigen, dass adaptive Reglerstrukturen stabilisierend auf Systemänderungen wirken. Dies entspricht der bisherigen Vorgehensweise im Rahmen der adaptiven nichtlinearen Regelung. Die Frage, welche Fehlereffekte von den künstlichen neuronalen Netzen genau erlernt werden können und wo die praktischen Grenzen des Erlernbaren liegen, wird dabei noch nicht beantwortet. Gleichzeitig zeigen erste Voruntersuchungen bereits, dass neuronale

Netze, z.B. in der Ebene der Lageregelung oder adaptive Aktuatormodelle, die Flugeigenschaften im Fehlerfall weiter verbessern können. Betrachtet man dazu die Vielzahl an Rekombinationsmöglichkeiten (Lernverfahren, Navigationsalgorithmen, Regler-Navigations–Strukturen), wird eine enger verzahnte Zusammenarbeit der Forschungsinstitute vorgeschlagen:

So war die Erprobung von Algorithmen in Flugversuchen bisher oft nur mit der Implementierung von Algorithmen-Code auf einer institutseigenen Hardware möglich. Die Wiederverwendbarkeit des Codes ist mit den Bibliotheken der Hardware verknüpft und stark von der Dokumentation abhängig. Die Einarbeitung neuer Programmierer in Code zur Wiederverwendung von Algorithmen ist sehr aufwendig und führt oft zu einer Neuentwicklung eines Algorithmus. Die Reproduzierbarkeit und Vergleichbarkeit der Flugversuchsergebnisse steht im Zusammenhang mit der Verfügbarkeit des Flugzeugmodells.

Die Möglichkeit der Nutzung von „Spielzeug“-Modellflugzeugen und Smartphonetechnik bietet hier eine kostengünstige Möglichkeit von einheitlichen Flugversuchen und des Vergleichs von Flugeigenschaften der adaptiven Regler-Navigations-Strukturen. Zu diesem Zweck wird vorgeschlagen, einheitliche UAV-Testplattformen als Referenzflugzeuge zu definieren.

Betrachtet man nun die Entwicklungen anderer Forschungsbereiche, z.B. den Bereich der Bildverarbeitung, so wird deutlich, dass die Rekombination verschiedener Algorithmen aus einer Bibliothek neue und unerwartete Möglichkeiten bietet. So ist z.B. die Erkennung markanter Bildpunkte allein ohne tieferen Wert. Kombiniert man diese allerdings mit Filtertechniken, einem Kalman Filter und Suchalgorithmen, wird die Gesichtserkennung möglich.

Die Verfügbarkeit von direkt programmierbaren Schaltplanbibliotheken im Themengebiet der adaptiven Regelung und die Möglichkeit der Nutzung einheitlicher Flugzeugmodelle hat das Potenzial, die Forschungs- und Entwicklungsgeschwindigkeit auf einer höheren Implementierungsebene, hin zu neuen Anwendungsbereichen, deutlich zu beschleunigen.

Mit steigender Komplexität der Algorithmen kommt man an die Grenzen eines sequenziellen Prozessors für Echtzeitanwendungen. Dennoch bietet der verwendete Prozessor genügend Ressourcen für weitere neuronale Netze, wie anhand von Grafik 7.1 dargestellt wird. Es ist absehbar, dass hier ab einer Anzahl von 6 neuronalen Netzen die Grenze der OMAP-3530 CPU-Ressourcen ausgeschöpft sind.

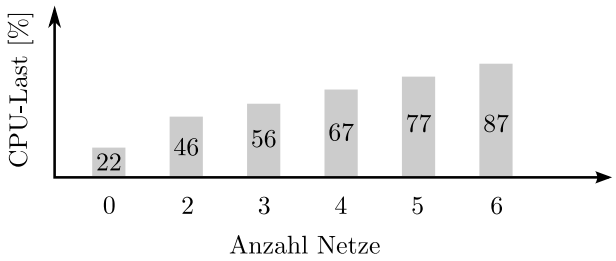


Abbildung 7.1.: CPU-Last bei dynamischer Inversion und Anzahl neuronaler Netze auf dem OMAP-3530 Prozessor

Die Entwicklung der letzten Jahre zeigt, dass der Kerntakt eines Desktop-CPU etwa bei 5 GHz eine Konvergenzlinie erreicht und diese kaum überschreiten wird. Zu hohe Verlustleistungen und Hochfrequenzeffekte scheinen eine physikalische Barriere zu bilden. Das bedeutet aber nicht, dass die Rechenleistung der Rechner konvergiert. Im Gegenteil zeichnet sich bereits ein Trend zu mehreren Rechenkernen mit Dual-, Quad- und Achtkernprozessoren ab. Mehrere Rechen-Kerne bedeuten nicht zwangsweise die Verfügbarkeit von mehr Rechenleistung, da bedingt durch die Architektur eines Posix-Thread, ein Thread immer sequentiell auf einem Kern abläuft. Einzig die Aufteilung von Vektor und Matrix-Operationen in mehrere Threads bzw. parallele Rechenschritte erlaubt ein beinahe unendliches Vergrößerungspotenzial in der Komplexitätssteigerung von Algorithmen. Dies bedeutet für die praktische Anwendung: Die Entwicklungsumgebungen müssen den Trends der Prozessorentwicklung folgen, um die Fähigkeiten mehrerer Rechenkerne effektiv für Echtzeitanwendungen nutzen zu können. Echtzeitbetriebssysteme sowie Programmiersprachen müssen einen Weg finden, die Rechenoperationen effektiv auf die Kerne zu Verteilen. Mit z.B. VHDL gelingt dies bereits auf FPGA's, mit CUDA auf Grafikkarten. Die Umsetzung für objekt- bzw. piktogrammorientierte Programmiersprachen wie Simulink in Kombination mit Autocode und Echtzeit bleibt eine Herausforderung.

Literaturverzeichnis

[Ada09] Adamy, J.: *Nichtlineare Regelungen.* Vierte Edition. Springer, ISBN-13: 978-3-642-00793-4, 2009

[Ana11] Analog, Devices: *Gyro Mechanical Performance: The Most Important Parameter.* Analog Devices, Technical Article MS-2158, 2011

[Ang09] Angner, S.: *Linux Realtime-Fähigkeiten.* Bachelor Diplomarbeit, Hochschule Luzern, 2009 `http://www.agner.ch/linuxrealtime/`. – [letzter Zugriff 19-10-2014]

[BAL11] Brockhaus, R. ; Alles, W. ; Luckner, R.: *Flugregelung.* Springer, 2011. – ISBN ISBN 978–3–642–01443–7

[Ban89] Banta, L.: *A Self-Tuning Navigation Algorithm For a Robot Vehicle.* American Control Conference, Pages 503-506, IEEE, 1989

[Bau07] Bauer, A.: *Realtime capabilities of low-end PowerPC and ARM boards for embedded systems.* Real-Time-Linux Workshop, Linz, Austria, 2007

[BI84] Byrnes, I. C. ; Isidory, A.: *A frequency domain philosophy for nonlinear systems.* Proc. 23rd IEEE Conf. Decision Contr., Las Vegas, NV, 1569-1573, 1984, 1984

[BL12] Bancroft, J. B. ; Lachapelle, G.: *Estimating MEMS Gyroscope G-Sensitivity Errors in Foot Mounted Navigation.* IEEE, 2nd Conference on Ubiquitous Positioning, Indoor Navigation and Location-Based Service, Helsinki, 2-5 Oct., 2012

[BS89] Bogacki, P. ; Shampine, F. L.: *A 3(2) Pair of Runge Kutta Formula.* Appl. Math. Lett. Vol. 2., No.4, pp. 321-325, 1989

[Cal96] Calise, J. A.: *Neural Networks in Nonlinear Aircraft Flight Control.* In: IEEE Aerospace and Electronics Systems Magazine, ISBN: 0-7803-2636-9, 1996

[CB05] Coggan, S. J. ; Bartol, M. T.: *Evidence for Ectopic Neurotransmission at a Neuronal Synapse.* Science, Vol. 309 no 5733 pp. 446-451, 2005

[CHN00] Calise, J. A. ; Hungu, L. ; Nakwan, K.: *High Bandwith Adaptive Flight Control.* In: IEEE Aerospace and Electronics Systems Magazine, ISBN: 0-7803-2636-9, 200

[CHU06] CHU, PONG P.: *RTL Hardware Design Using VHDL.* John Wiley & Sons, Inc., Hoboken, New Jersey. ISBN-13: 978-0-471-72092-8, 2006

[CJC+13] Chowdhary, G. ; Johnson, N. E. ; Chandramohan, R. ; Kimbrell, S. M. ; Calise, A.: *Autonomous Guidance and Control of Airplanes under Actuator Failures and Severe Structural Damage.* Georgia Institute of Technology, Atlanta, AIAA Journal of Guidance, Control, and Dynamics, Vol. 36, No. 4, pp. 1093-1104, 2013

[CPN06] Christophersen, B. H. ; Pickell, W. R. ; Neidhoefer, C. J.: *A Compact Guidance, Navigation, and Control System for Unmanned Aerial Vehicles.* Journal of Aerospace Computing, Information, And Communication Vol. 3, 2006

[Cra13] Crassidis, L. J.: *Sigma-Point Kalman Filtering for Integrated GPS and Inertial Navigation.* University at Buffalo, State University of New York, Amherst, NY 14260-4400, 2013

[CW03] Congwei, H. ; Wu, C.: *Adaptive Kalman Filtering for Vehicle Navigation.* Journal of Global Positioning Systems, Vol. 2, No. 1: 42-47, 2003

[DAL13] Dydek, Z. T. ; Annaswamy, A. M. ; Laveretsky, E.: *Adaptive Control of Quadrotor UAVs.* IEEE Transaction on Control Systems Technology, Vol. 21, No. 4, 2013

[Deh83] Dehn, C.: *Ein Beitrag zum Einsatz von Mikrorechnern in Trägheitsnavigationssystemen mit fahrzeugfesten Sensoren und zur Stützung von Mess-Systemen mittlerer Genauigkeit.* TU Braunschweig, 1983. – Diss.

[DMC00] Drolet, L. ; Michaud, F. ; Cote, J.: *Adaptable Sensor Fusion Using Multiple Kalman Filters.* International Conference on Intelligent Robots and Systems, IEEE, 2000

[DP77] Dormand, J.R. ; Prince, P.J.: *New Runge-Kutta Algorithms for Numerical Simulation in Dynamical Astronomy.* Department of Mathematics and Statistics, Teesside Polytechnics, Cleveland, 1977

[Dra00] Draghici, S.: *Neural Networks in Analog Hardware - Design and Implementation Issues.* Wayne State University, Detroit USA, Department of Computer Science, 2000

[DY10] Domont-Yankulova, D.: *Entwurf strukturvariabler Regelungen mittels linearer Matrixungleichungen.* Technische Universität Darmstadt, Diss, 2010

[EK02] Efe, M. ; Kaynak, O.: *Lecture Notes in Control and Information Sciences. Bd. 274: Variable Structure Systems: Toward the 21st Century. Variable Strucutre Systems Theory in Computational Intelligence.* Springer Verlag, Berlin, 2002

[Eme69] Emelyanov, S. V.: *Automatische Regelsysteme mit veränderlicher Struktur.* Oldenburg, München, 1969

[Eng95] Engell, S.: *Entwurf nichtlinearer Regelungen.* R. Oldenbourg Verlag München Wien, ISBN: 978-3486230659, 1995

[ES98] Edwards, C. ; Spurgeon, S. K.: *Sliding Mode Control, Theory and Applications.* CRC Press, ISBN-13: 978-0748406012, 1998

[Fer09] Fernandez, M. I L.: *Design, Implementation and Flight Verification of a Versatile and Rapidly Reconfigurable UAV GNC Research Platform.* University of California, Santa Cruz, 2009

[FLT55] Függe-Lotz, I. ; Taylor, C.: *Synthesis of a nonlinear control system.* WESCON, West Cost Conference, 1955

[Gup09] Guppa, S. D.: *A Comparitive Study of the Particle Filter and the Ensemble Kalman Filter.* Master Thesis, University of Waterloo, Ontario, Canada, 2009

[Heb95] Hebisch, H.: *Grundlagen der Sliding-Mode-Regelung.* Gerhard-Mercator-Universit Duisburg, 1995

[HNW93] Hairer, E. ; Norsett, P. S. ; Wanner, G.: *Solving Ordinary Differential Equations.* Springer Verlag, ISBN 978-3-540-78862-1, 1993

[Hol04] Holzapfel, F.: *Nichtlineare adaptive Regelung eines unbemannten Fluggerätes.* Technische Universität München, Diss., 2004

[HSB11] Hanson, C. ; Schaefer, J. ; Burken, J. J.: *Handling Qualities Evaluations of Low Complexity Model Reference Adaptive Controllers for Reduced Pitch and Roll Damping Scenarios.* AIAA Guidance, Navigation, and Control Conference, 2011 `http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20110023802.pdf`. – [letzter Zugriff 2-12-2014]

[Isi95] Isidory, A.: *Nonlinear Control Systems.* 3rd edition. Springer, ISBN-13: 978-3540199168, 1995

[IWH95] Irwin, W. G. ; Warwick, K. ; Hunt, K. J.: *Neural Networks Applications in Control.* The Institution of Electrical Engineers, United Kingdom, 1995

[JC00] Johnson, N. E. ; Calise, J. A.: *Pseudo-Control Hedging: A New Method for Adaptive Control.* Advances in Navigation and Control Technology Workshop, Redsonte Arsenal, Alabama, 2000

[JL01] Joos, H.-D. ; Looye, G.: *Design of Robust Dynamic Inversion Control Laws using Multi-Objective Optimization.* AIAA-2001-4285. AIAA Guidance, Navigation, and Control Conference and Exhibit, Montreal, Kanada, 2001

[Joh00] Johnson, N. E.: *Limited Authority Adaptive Flight Control.* Georgia Institute of Technology, Diss., 2000

[Kam11] Kammerer, R.: *Linux in Safety-Critical Applications (OSADL Academic Works, Volume 1).* Open Source Automation Development Lab (OSADL) eG; 1ST edition, ISBN-13: 978-3000338854, 2011

[KB12] Kroonenberg, A. ; Bange, J.: *Spatially-Averaged Temperature Structure Parameter Over a Heterogeneous Surface Measured by an Unmanned Aerial Vehicle.* In: Boundary-Layer Meteorology 142 (2012), S. 55Ű77, 2012

[KEE01] Kaynak, O. ; Erbatur, K. ; Ertugrul, M.: *The Fusion of Computationally Intelligent Methodologies and Sliding-Mode Control - A Survey.* Transactions on Industrial Electronics 48, IEEE, 2001

[Kha02] Khalil, H. K.: *Nonlinear Systems.* Prentice Hall, NJ, 2002. – ISBN 0–13–067389–7

[Kit12] Kittmann, K.: *Entwicklung einer modularen Messplattform zur Analyse des Potenzials von Freiflugmessungen.* Institut für Flugzeugbau der Universität Stuttgart, 2012

[KK14] Krüger, A. ; Kufieta, K.: *Ein verteiltes Autopilotensystem für Forschung und Lehre.* 63.ter Deutscher Luft- und Raumfahrtkongress, Augsburg, 2014

[Krü12] Krüger, T.: *Zur Anwendung neuronaler Netzwerke in adaptiven Flugregelungssystemen.* Technische Universität Braunschweig, Diss., ISBN: 9783832532505, 2012

[Kra13] Krause, S.: *Entwicklung und Aufbau eines Messsystems zur Bestimmung von Steuerflächenausschlägen.* Technische Universität Braunschweig, Institut für Luft- und Raumfahrtsysteme, Studienarbeit, 2013

[Kuf09] Kufieta, K.: *Aufbau einer Quadrocopter Architektur und Erweiterung um eine Realtime Workshop Autocode Plattform.* Studienarbeit, Institut für Systemdynamik, Universität Stuttgart, 2009

[KWR+10] Krüger, T. ; Wilkens, C.-S. ; Reinhold, M. ; Selsam, P. ; Böhm, B. ; Vörsmann, P.: *Ergebnisse des ANDROMEDA-Projektes - Automatische Luftbildgewinnung mit Unbemannten Kleinflugzeugen.* In: Deutscher Luft- und Raumfahrtkongress. Hamburg, PaperID 161314, 2010. – ISBN ISBN 978–3–642–01443–7

[KWV11] Krüger, T. ; Wilkens, C.-S. ; Vörsmann, P.: *Automatische Luftbildgewinnung mit Unbemannten Kleinflugzeugen.* In: German Aeronautics and Astronautics Congress. Hamburg, Germany, September 2010. ISBN:978-3-932182-68-5, PaperID 161314, 2011

[Lan13] Landa, T.: *Untersuchung neuronaler Netzwerke als Approximator in einem adaptiven Flugregelungssystem.* TU Braunschweig, 2013

[Lin90] Lindemann, U.: *Entwurf eines selbsteinstellenden, mehrachsigen Flugreglers mit lernender Vorsteuerung.* Technische Universität Braunschweig, Diss., 1990

[LL67] LaSalle, J. ; Lefschetz, S.: *Die Stabilitätsanalyse nach Ljapunow.* Bibliographisches Institut, Mannheim, 1967

[Loo08] Looye, G.: *An Integrated Approach to Aircraft Modelling and Flight Control Law Design.* Technische Universität Delft, ISBN: 978-90-5335-148-2, 2008

[Lor10] Lorenz, S.: *Adaptive Regelung zur Flugbereichserweiterung des Technologiedemonstrators ARTIS.* Deutsches Zentrum für Luft- und Raumfahrt, Insitut für Flugsystemtechnik Braunschweig, Diss., 2010

[LYL96] Lewis, F. L. ; Yegildirek, A. ; Liu, K.: *Multilayer Neural-Net Robot Controller with Guaranteed Tracking Performance.* IEEE Transactions on Neural Networks, Vol.7 No.2, S. 388-399, 1996

[Mac04] Macke, O.: *Entwicklung eines Konzeptes zur Bahnführung von Mikroflugzeugen auf gekrümmten Bahnen.* Technische Universität Braunscheig, Institut für Luft- und Raumfahrtsysteme, Studienarbeit, 2004

[Man01] Mancini, R.: *Op Amps for Everyone.* Texas Instruments, Design Reference, 2001

[MB11] Martin, S. ; Bange, J.: *Meteorological profiling of the lower troposphere using the research UAV „M2AV Carolo“.* In: Atmos. Meas. Tech. 4 (2011),S. 705Ű716, 2011

[Möß09] Mössner, M.: *Optimierung einer adaptiven neuronalen Reglerstruktur und Stabilitätsanalyse der verwendeten Lernverfahren.* Technische Universität Braunscheig, Studienarbeit, 2009

[Ngu06] Nguyen, V. M. T.: *Advanced Neural Network Controllers and Claccifiers Based on Sliding Mode Training Algorithms.* The University of Technology, Sydney, Diss., 2006

[Now10] Nowack, J.: *Windkanal-Freiflugmessungen zur Bestimmung flugmechanischer Kenngrößen.* Fakultät für Maschinenwesen der Rheinisch-Westfälischen Technischen Hochschule Aachen, Diss., 2010

[NS07] Nguyen, N. T. ; Stephen, J. A.: *Neural Net Adaptive Flight Control Stability, Verification and Validation Challenges, and Future Research.* IJCNN Conference, 2007

[OR06] Omondi, A. R. ; Rajapakse, J. C.: *FPGA Implementations of Neural Networks.* Springer, ISBN-13 978-0-387-28487-3 (e-book), 2006

[OSA14] OSADL: *Latency Plots OMAP 3525.* Open Source Automation Developement Lab (OSADL), 2014 `https://www.osadl.org/Hardware-overview.qa-farm-hardware.0.html`. – [letzter Zugriff 11-10-2014]

[Pfe14] Pfeifer, T.: *Dynamische Inversion im Flugversuch.* Technische Universität Braunschweig, Institut für Luft- und Raumfahrtsysteme, Studienarbeit, 2014

[Pla10] Placzek, R.: *Analyse und Optimierung neuronaler Elemente in adaptiven Regelungsverfahren.* Technische Universität Braunschweig, Studienarbeit, 2010

[PM98] Parma, G. G. ; Menezes, R. B.: *Sliding mode algorithm for training multilayer artificial neural networks.* Electronics Letters Vol.34 No.1, IEEE, 1998

[PMBB98] Parma, G. G. ; Menezes B., R. ; Braga, P. A.: *Sliding mode backpropagation: control theory applied to neural network learning.* In: Proc. 'Int. Joint Conf. Neural Networks IJCNN '99 Bd. 3, 1774-1778, 1998

[RA13] Reimann, M. ; Anastassiou, C.: *A biophysically detailed model of neocortical local field potentials predicts the critical role of active membrane currents.* Blue Brain Project, École Polytechnique Fédérale de Lausanne, 2013

[RBB09] Richard, L. R. ; Burken, J. J. ; Butler, B. S.: *Implementation of an Adaptive Controller System from Concept to Flight Test.* NASA Dryden Flight Research Center Edwards, California, 2009

[RHW86] Rummelhart, D. E. ; Hinton, G. E. ; Williams, R. J.: *Learning representations by back-propagating errors.* Nature, Vol. 323, No 6088 pp475-566, 1986

[Rob13] Robertz, C.: *Optimierung eines adaptiven Flugregelkreises zur Erprobung im Flugversuch.* Diplomarbeit TU Braunscheig, 2013

[Roj96] Rojas, R.: *Neural Networks.* Springer Berlin, 1996

[Sch02] Schulz, H.-W.: *Aufbau und Durchführung von Versuchen zur Vermessung von Modellbau-Servos und Ermittlung theoretischer Modelle zur*

Beschreibung des dynamischen Übertragungsverhaltens. Technische Universität Braunscheig, Institut für Luft- und Raumfahrtsysteme, Diplomarbeit, 2002

[Sch08] Schulz, H.-W.: *Ein rekonfigurierbares Bildverarbeitungssystem für unbemannte Kleinstflugzeuge - Entwicklung eins bildgestützten Landeverfahrens.* Technische Universität Braunscheig, Institut für Luft- und Raumfahrtsysteme, Diss., 2008

[Sch11a] Scheck, M.: *Anwendung von stabilen Sliding-Mode Trainingsverfahren auf neuronale Regler.* Studienarbeit, Technische Universität Braunschweig, 2011

[Sch11b] Schnetter, P.: *Entwicklung eines Flugreglers auf Basis dynamischer Inversion.* Diplomarbeit TU Braunschweig, 2011

[Sch13] Scholz, G.: *Erweiterung eines adaptiven Flugreglers zur Kompensation begrentzter Aktuatordynamik.* Technische Universität Braunschweig, Institut für Luft- und Raumfahrtsysteme, Diplomarbeit, 2013

[SCT15] Scholz, G. ; Crocoll, P. ; Trommer, G. F.: *Nonlinear Inverse Dynamics used to Control a Quadrocopter with Tiltable Rotors.* Proceedings of the 2015 International Technical Meeting of The Institute of Navigation, Dana Point, California, January 2015, pp. 745-755., 2015

[SGL10] Schumann, J. ; Gupta, P. ; Liu, Y.: *Application of Neural Networks in High Assurance Systems: A Survey.* Springer-Verlag Berlin Heidelberg, 2010

[SH89] Stinchcombe, M. ; Hornik, K.: *Mulitlayer Feedforward Networks are Universal Approximators.* Neural Networks, Vol.2 pp. 359-366, 1989

[SIK87] Sarpturk, S. ; Istefanopulus, Y. ; Kaynak, O.: *On the Stability of Discrete-Time Sliding Mode Control Systems.* In: IEEE Transactions on Automatic Control AC-32, No. 10, October, 1987

[Sim11] Simplicio, P.: *Helicopter Nonlinear Flight Control using Incremental Nonlinear Dynamic Inversion.* Universidade Tecnica de Lisbo, Diss., 2011

[Smi97] Smith, S. W.: *The Scientist and Engineer's Guide to Digital Signal Processing*. Bertrams ISBN-13: 978-0966017632, 1997 `http://www.dspguide.com/ch28/5.htm`. – [letzter Zugriff 22-11-2014]

[SW91] Slotine, J. ; Weiping, L.: *Applied Nonlinear Control*. Pretince Hall, Englewood Cliffs, NJ, 1991

[Tan02] Tannenbaum, A. S.: *Moderne Betriebssysteme*. Pearson Studium. ISBN-13: 978-3827370198, 2002

[TK01] Topalov, A. V. ; Kaynak, O.: *Online Learning in Adaptive Neurocontrol Schemes with a Sliding Mode Algorithm*. In: IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics 31, Nr. 3, S. 445-450, 2001

[u13] u blox: *LEA 6 Datasheet*. u-Blox, 2013 `http://www.u-blox.com/images/downloads/Product_Docs/LEA-6_DataSheet_(GPS.G6-HW-09004).pdf`. – [letzter Zugriff 20-10-2014]

[Utk78] Utkin, V. L.: *Sliding Modes and Their Application in Variable Structure Systems*. Imported Publications, Incorporated, 1978. – ISBN 9780828506960

[Utk92] Utkin, V. L.: *Sliding Modes in Control and Optimization*. Springer Verlag, Berlin, 1992

[Win06] Winkler, S.: *Zur Sensordatenfusion für integrierte Navigationssysteme unbemannter Kleinstflugzeuge*. Technische Universität Braunschweig, Institut für Luft- und Raumfahrtsysteme, 2006

[WK00] Wang, J. ; Kusiak, A.: *Computational Intelligence in Manufacturing Handbook*. CRC Press, 2000

[XJXU02] Xinghuo, Y. ; Jian-Xin, X. ; Utkin, V.: *Variable Structure Systems: Towards the 21st Century*. Springer Verlag, Berlin, 2002

[YK09] Yu, Xinghuo ; Kaynak, O.: *Sliding-Mode Control With Soft Computing: A Survey*. Transactions on Industrial Electronics 56, IEEE, 2009

[YW11] Yu, Hao ; Wilamowski, B. M.: *Levenberg-Marquardt Training*. Industrial Electronics Handbook, vol. 5 - Intelligent Systems, 2nd Edition, chapter 12, pp. 12-1 to 12-15, CRC Press 2011, 2011

A Simulation KNN-NDI mit E-Modifikation, Wind und Sensorrauschen

Die folgende Simulation entspricht der dynamischen Inversion mit PCH und Sliding-Mode-trainierten Netzen mit E-Modifikation. Simuliert wird dabei ein konstanter Wind mit $\vec{V_c} = [1\ \text{m/s}, 1\ \text{m/s}, 0]^T$, Turbulenzen mit $\sigma = 3$ (Kapitel 3.1) und die Sensorfehler aus Kapitel 2.2.

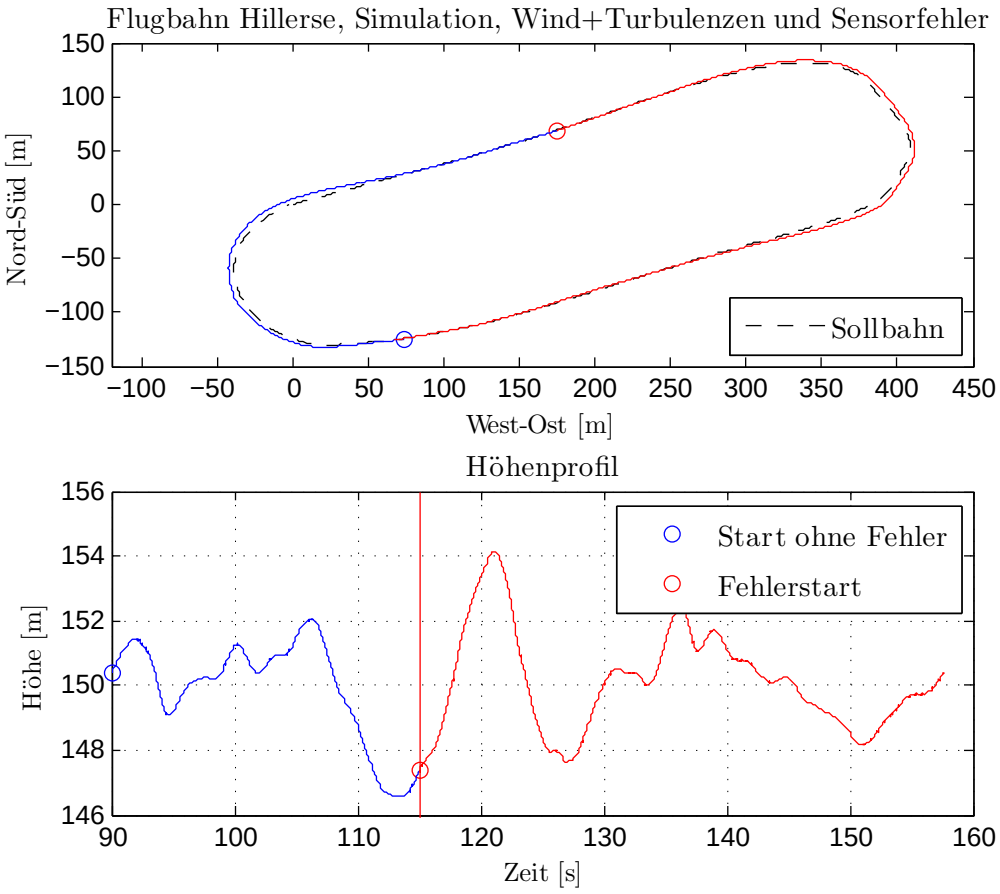


Abbildung A.1.: Simulation mit neuronalen Netzen; Flugbahn und Flughöhe; rote vert. Linie: blockiertes l. Querruder $\xi_{li} = 7°$.

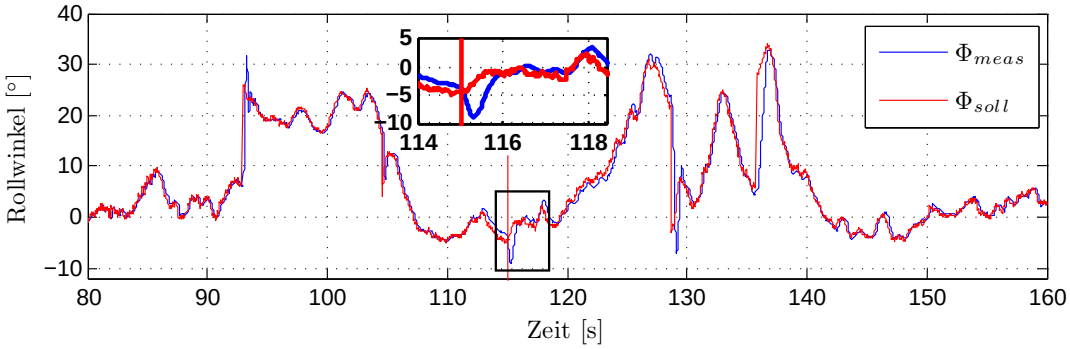


Abbildung A.2.: Simulation mit neuronalen Netzen; gemessene und kommandierte Nickrate; rote vert. Linie: blockiertes l. Querruder $\xi_{li} = 7°$.

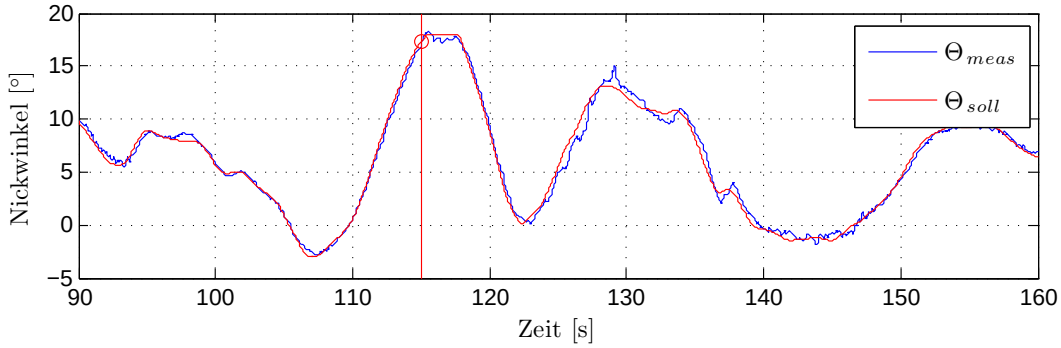


Abbildung A.3.: Simulation mit neuronalen Netzen; gemessene und kommandierte Nicklage; rote vert. Linie: blockiertes l. Querruder $\xi_{li} = 7°$.

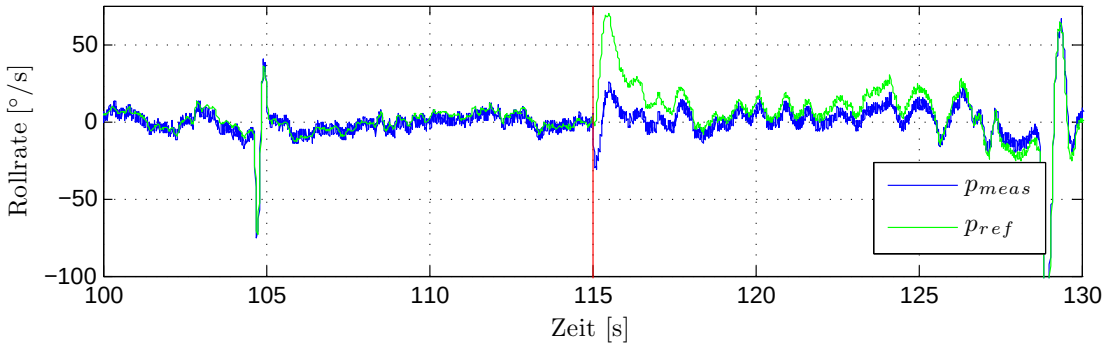


Abbildung A.4.: Simulation mit neuronalen Netzen; gemessene Rollrate p_{meas} und Referenzsignal Rollrate p_{ref}; rote vert. Linie: blockiertes l. Querruder $\xi_{li} = 7°$.

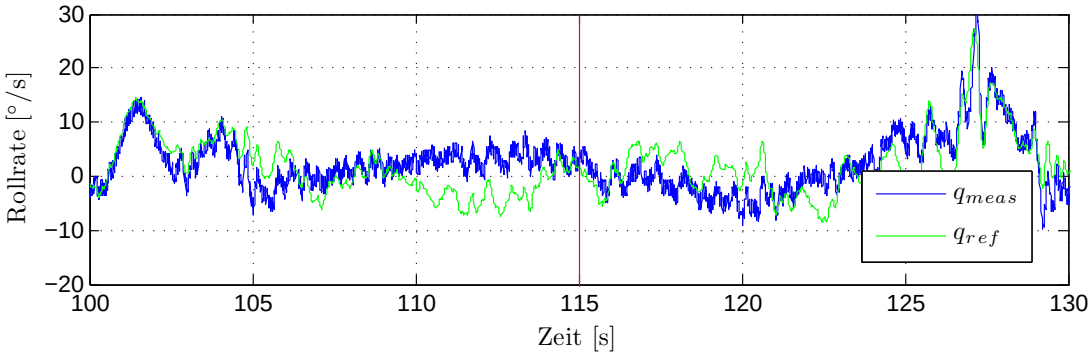


Abbildung A.5.: Simulation mit neuronalen Netzen; gemessene Nickrate q_{meas} und Referenzsignal Rollrate q_{ref}; rote vert. Linie: blockiertes l. Querruder $\xi_{li} = 7°$.

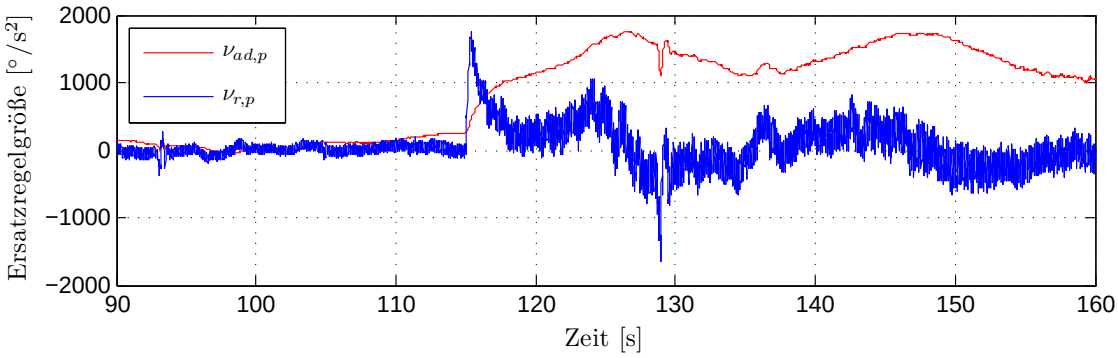


Abbildung A.6.: Simulation mit neuronalen Netzen; Ausgang neuronale Netze $\nu_{ad,p}$ und robustifizierender Term $\nu_{r,p}$; Querruderfehler ab [t=115 s]; ab Fehler Skalierung für Einheit [$°/s^2$] ungültig.

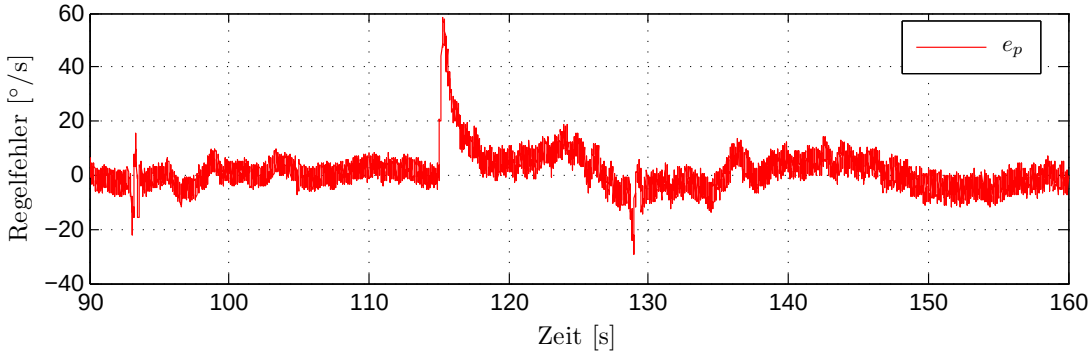


Abbildung A.7.: Simulation mit neuronalen Netzen; Regelfehler e_p; Querruderfehler ab [t=115 s].

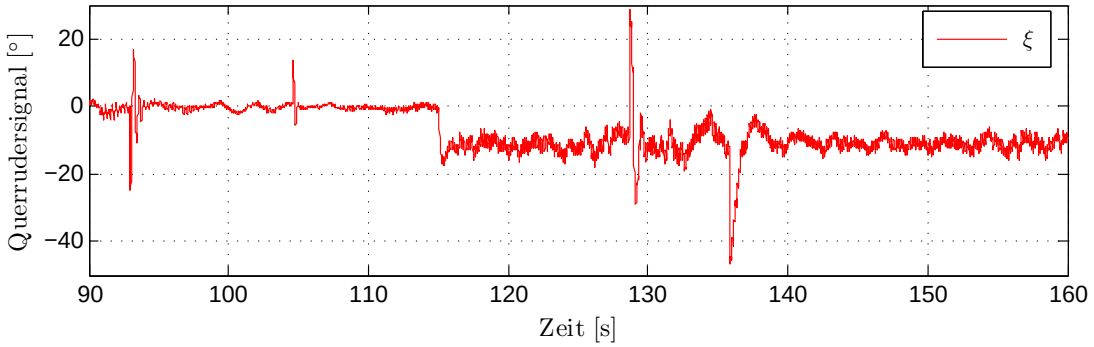


Abbildung A.8.: Simulation mit neuronalen Netzen; Querrudersignal; Querruderfehler ab [t=115 s].

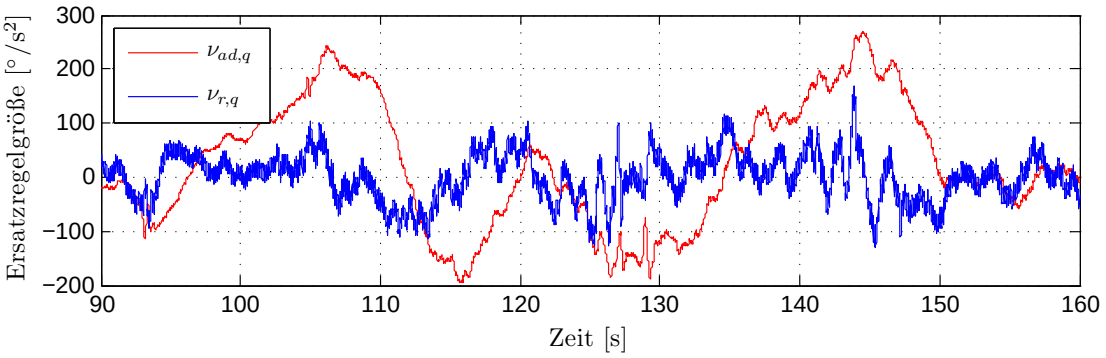


Abbildung A.9.: Simulation mit neuronalen Netzen; Ausgang neuronale Netze $\nu_{ad,q}$ und robustifizierender Term $\nu_{r,q}$; Querruderfehler ab [t=115 s].

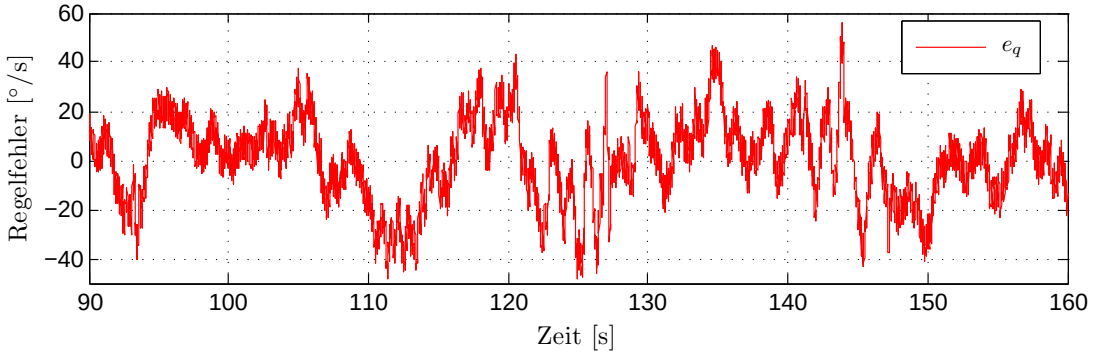


Abbildung A.10.: Simulation mit neuronalen Netzen; Regelfehler e_q; Querruderfehler ab [t=115 s].

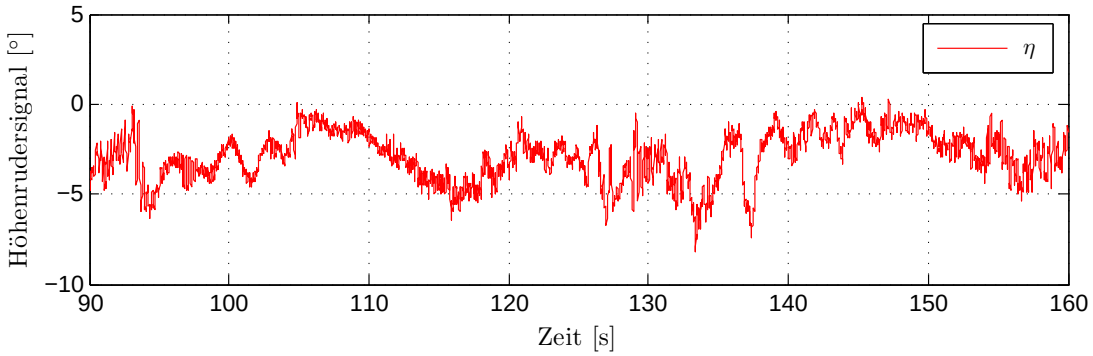


Abbildung A.11.: Simulation mit neuronalen Netzen; Höhenrudersignal; Querruderfehler ab [t=115 s].